UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Sistemas de Información

Trabajo Fin de Grado

Diseño e Implementación de una herramienta visual que permita evaluar expresiones del álgebra relacional

Autor: Daniel Magallón Martínez

Tutor: Iván González Diego

TRIBUNAL

Presidente:

Vocal 1º:

Vocal 2º:

FECHA:

Agradecimientos

En primer lugar quisiera agradecer a Iván González Diego la oportunidad que me ha brindado para realizar este proyecto y por su dedicación y tiempo cuando ha sido necesario, el cual ha sido imprescindible para poder realizarlo.

Me gustaría agradecer también a mi familia por su apoyo brindado a lo largo de este proceso, en los buenos y en los malos momentos, en especial a mis padres que han estado siempre ahí para lo que fuese necesario.

Agradecer también a todos mis compañeros y amigos de la universidad, sin los cuales seguramente no podría haber llegado hasta aquí. Por ello y por lo momentos vividos quería agradecerles todo este camino que hemos recorrido juntos.

Por último agradecer a todos los profesores que me han impartido clase a lo largo del Grado, por su dedicación y trabajo.

Gracias.

Tabla de contenido

Tabla de contenido

Introducción

El objetivo de este capítulo es proporcionar una visión general del proyecto. A lo largo de la introducción daré una visión objetiva de lo que compondrá el mismo mostrando un resumen, el contexto, la motivación y los objetivos del proyecto realizado. Indicaré también otros aspectos relacionados con el proyecto como pueden ser la metodología y los recursos utilizados para la realización del mismo.

Resumen

El proyecto consiste en la elaboración de una aplicación que sea capaz de transformar cualquier consulta de Álgebra Relacional en una consulta SQL equivalente. La herramienta deberá conectarse con cualquier base de datos relacional en PostgreSQL.

La herramienta está pensada para un uso didáctico posterior hacia los nuevos alumnos de la UAH que comiencen su aprendizaje con bases de datos relacionales y con el lenguaje del Álgebra relacional. Es una herramienta que ahora mismo no se encuentra en el mercado y cuyo uso puede ser beneficioso para el aprendizaje de los nuevos alumnos.

Summary

The project consists in making an application that will be able to transform any query about Relational Algebra on a SQL query equal. The app must connect with any relational database in PostgreSQL.

The application has been designed to an educational use for the new students of UAH that begin their learning with relational databases and relational algebra language. This is an app that is not in market at this moment and can be useful to the learning of the new students.

Contexto

Actualmente en el mundo corporativo todas las empresas hacen uso de bases de datos para organizar la información, y prácticamente la mayoría de ellas hacen uso del modelo relacional. En dicho modelo, basado en las relaciones entre los elementos, se pueden realizar distintas consultas para añadir, eliminar, modificar o consultar datos de las mismas.

Existen distintos lenguajes de consulta entre los que vamos a destacar en este caso el SQL y el álgebra relacional. El SQL deriva de un lenguaje matemático denominado álgebra relacional del cual derivan muchos de los operadores del lenguaje SQL y su forma de funcionamiento. El software que se quiere implantar realizará una transformación de una consulta en álgebra relacional (ininteligible para el SGBD) a una consulta en lenguaje SQL, para su posterior ejecución en una base de datos PostgreSQL y retorno de resultados.

La herramienta esta ideada para un uso didáctico posterior. Cuando se comienza a tratar con bases de datos el primer lenguaje que aprendes es el SQL, debido a su simpleza y potencia. Este lenguaje se puede utilizar directamente en una sentencia reconocible para el sistema gestor de base de datos y que éste pueda realizar las acciones que se desea. Posteriormente se trata el álgebra relacional, un lenguaje con gran variedad de símbolos y mayor dificultad para su entendimiento que el lenguaje SQL. Aparece un problema al tratar con este lenguaje, los SGBD utilizados a lo largo del Grado no pueden ejecutar sentencias generadas de este modo.

Debido a este problema se ha decidido definir e implantar una nueva herramienta que sea capaz de recoger una sentencia en Álgebra Relacional introducida manualmente por el usuario, para después transformarla a lenguaje SQL y ejecutar la sentencia en una base de datos PostgreSQL determinada. La herramienta contará con botones que representen los distintos símbolos de este lenguaje, esta serie de botones servirán para construir sentencias del álgebra relacional y una pantalla donde aparecerán los resultados de las búsquedas.

Este software permitirá a los alumnos lograr una mayor comprensión del Álgebra Relacional y les ofrecerá una herramienta que les permita practicar con ella de forma sencilla, sobre todo para la asignatura de tercero para la optimización de consultas.

Álgebra relacional

Uno de los primeros aspectos a tratar es en qué consiste el álgebra relacional y su impacto en el mundo actual. Como su nombre indica forma parte del modelo relacional de bases de datos, el cual vamos a tratar. El modelo relacional, debido a su simplicidad con respecto a otros modelos como el de red o el jerárquico, se ha posicionado como el principal modelo de datos para las aplicaciones de procesamiento de información, ya que ofrece una forma muy simple y potente de representar datos.

Una base de datos relacional es formada por un conjunto de tablas, a cada una de ellas se les asigna un nombre exclusivo. Cada tabla tiene una estructura parecida a las que aparecen en el modelo Entidad-Relación. Cada fila dentro de la tabla representa una relación entre un conjunto de valores. Aparece una importante correspondencia entre el concepto de tabla y el concepto matemático de relación, ya que cada tabla es un conjunto de las ya mencionadas relaciones.

El modelo relacional cuenta con tres lenguajes formales de consulta, el cálculo relacional de tuplas, el cálculo relacional de dominios y el álgebra relacional, siendo este último el que vamos a tratar ya que es el que he utilizado a lo largo del proyecto.

El álgebra relacional es un lenguaje de consulta procedimental en el cual un usuario solicita información de la base de datos. Está formado por un conjunto de operaciones que toman como entrada una o dos relaciones y generan como resultado una nueva relación. Este lenguaje consta de una serie de operaciones, fundamentales y no fundamentales.

Operaciones fundamentales:

- Selección: Permite seleccionar un subconjunto de tuplas de una relación (R), todas aquellas que cumplan la condición P.
- Proyección: Permite extraer columnas de una relación, dando como resultado un subconjunto vertical de atributos de la relación.
- Unión: Devuelve el conjunto de tuplas que están en R, o en S, o en ambas. R y S deben ser uniones compatibles.
- Diferencia de conjuntos: Devuelve todas las tuplas que están en R, pero no en S. R y S deben ser uniones compatibles.
- Producto cartesiano: Obtiene una combinación de todas las tuplas R con cada una de las tuplas de S.
- Renombramiento: Permite renombrar una subconsulta para utilizarla posteriormente.

Operaciones no fundamentales:

- Intersección de conjuntos: Obtiene todas las tuplas que están en R y en S. R y S deben de ser uniones compatibles.
- Reunión natural: También conocido como Natural Join. Permite combinar proyección, selección y producto cartesiano en una sola operación. La proyección elimina las columnas duplicadas.
- Asignación: Permite agrupar conjuntos de valores en función de un campo determinado y hacer operaciones con otros campos.

El álgebra relacional permite también acciones de inserción y borrado pero no serán tratadas a lo largo del trabajo ya que la herramienta solo esta ideada para consultas sobre la base de datos. Se trabajará con el álgebra relacional básica, ya que existe la extendida donde se definen además otros tipos de operadores.

Motivación

La principal motivación de la realización del proyecto que llevo a cabo es la de ayudar a los nuevos alumnos que comiencen a tratar con bases de datos relacionales. El comienzo de tratar con el lenguaje de Álgebra Relacional es un camino complejo, ya que es simbología abstracta y que no puedes comprobar resultados sobre ninguna base de datos. Por ello gracias a la aplicación se podrán realizar dichas comprobaciones lo cual será algo beneficioso y algo que yo hubiese agradecido en mis primeros pasos sobre las bases de datos.

Por otra parte, si el tutor estuviese de acuerdo, se podrían hacer algunas modificaciones posteriormente para conseguir otras funcionalidades interesantes. Por ejemplo sería mostrar la consulta traducida al lenguaje SQL, en vez de ejecutarla sobre la base de datos que simplemente sea mostrada una vez que la aplicación la traduzca. No se ha decidido hacer desde un principio porque la traducción no queda optimizada y puede ser un poco difícil de comprender. De todas formas para consultas simples puede utilizarse y ser muy beneficioso ya que uno de los aspectos más complejos de la asignatura de Bases de Datos Avanzadas es la de la traducción entre lenguajes.

Como ya se explicará a continuación no hay en el mercado, ni libre ni de pago, una aplicación que cumpla estos requisitos por lo que me pareció muy interesante su elaboración.

Objetivos del Proyecto

El objetivo principal del proyecto consiste en **definir e implantar una herramienta que sea capaz de transformar consultas de Álgebra Relacional en consultas SQL para posteriormente ejecutarlas en una base de datos relacional determinada.**

Este es el objetivo principal, del cual se pueden añadir o desgranar los siguientes objetivos específicos.

- Definir el lenguaje y las herramientas que se van a utilizar.
- Recopilación de información y estudio del lenguaje y herramientas a utilizar.
- Definir la estructura interna de la herramienta.
- Diseñar el aspecto visual, el cual debe de ser sencillo, visualmente claro y sencillo de manipular.
- Conexión entre la herramienta y el SGBD.

- Implementación de la herramienta y su código.
- Realización de pruebas de funcionamiento y búsqueda de errores.
- Posterior uso con finalidad didáctica.

Metodología

Lo primero que se ha hecho ha sido definir las herramientas con las que se iba a desarrollar la herramienta. Para el aspecto de la programación se ha decidido utilizar el lenguaje de programación Java, debido a su potencia y a la facilidad que ofrece para estos casos. Para la conexión entre la herramienta y el SGBD se ha decidido también utilizar el componente JDBC. Como sistema gestor de base de datos se va a utilizar PostgreSQL ya que es la herramienta que se ha utilizado a lo largo del Grado.

Se ha comenzado con una primera idea del código para conocer que estructura va a tener y qué posibles módulos iban a hacer falta. Antes de empezar a desarrollar el código se realizó la conexión con el componente jdbc para poder ir haciendo pruebas.

Por último, una vez terminada la herramienta, se han realizado las pruebas en busca de fallos, errores o incongruencias en el código. Una vez que se ha finalizado este aspecto he comenzado a desarrollar la memoria.

Recursos

Para la realización del proyecto se ha hecho uso de información buscada en Internet y bibliotecas, así como consulta de libros orientados a la programación y base de datos. También se ha recurrido a los apuntes de las asignaturas de programación y BBDD recogidos a lo largo del curso.

Independientemente de los recursos mencionados se ha necesitado el siguiente equipo informático:

- Hardware: Ordenador personal LENOVO Ideapad 100: procesador Intel® Core™ i5
- Software: Netbeans IDE 8.1, PostgreSQL 9.4, pgAdmin III, Microsoft Word 2013

Palabras clave

Algebra Relacional, SQL, Traductor, Bases de Datos, PostgreSQL.

Análisis

A continuación se va a realizar un análisis previo sobre la herramienta que se va a desarrollar. Este análisis incluirá aspectos variados que abarcará desde los requisitos previos hasta el prototipo.

Requisitos

Se necesita una aplicación que sea capaz de poder realizar los siguientes requisitos:

- Conexión entre la aplicación y una base de datos PostgreSQL cualquiera. La conexión se realizará mediante la autentificación tanto de usuario, contraseña y nombre de base de datos, así como el puerto y la IP que corresponden para la conexión.
- Introducción de consulta en álgebra relacional. La aplicación debe permitir mediante algún sistema poder introducir una consulta legible en el idioma de álgebra relacional.
- Transformación de consultas en álgebra relacional a consultas SQL. El sistema debe transformar dichas consultas a un idioma operativo para la base de datos con la que estamos trabajando.
- Devolución de resultados. Una vez transformada la consulta, debe de ser enviada al Sistema Gestor de Base de Datos para que pueda ejecutarla y devuelva los resultados. Estos resultados deben ser expuestos por la aplicación al usuario.

Análisis de requisitos

Una vez definidos los requisitos, se hará un análisis para estudiar cómo va a ser nuestra aplicación para que sea capaz de realizar dichos requisitos. Se tratarán las acciones y herramientas utilizadas para cumplir cada uno de los requisitos expuestos.

- Conexión entre la aplicación y una base de datos PostgreSQL: La conexión se ha hecho a cargo del componente jdbc, el cual es un complemento del software de desarrollo NetBeans. Jdbc nos permite establecer una comunicación entre nuestra aplicación y el SGBD PostgreSQL. Para realizar la conexión el usuario debe introducir los datos de su base de datos, estos son: Usuario, contraseña, nombre de BBDD, puerto y host. Cada vez que se inicie el programa se podrá conectar a una base de datos distinta cualquiera el usuario.

Hablando en términos de programación, se ha creado un objeto en el cual se engloban otros dos objetos, uno de consulta y otro de conexión. El objeto de consulta contiene la consulta introducida por el usuario ya traducida al idioma SQL, y el objeto de conexión contendrá todos los datos necesarios para poder realizar la conexión con el sistema gestor.

Primero se envía el objeto de conexión para comprobar que la conexión se realiza correctamente. Si esto es así se avanzará a la siguiente pantalla sabiendo que los datos introducidos son correctos.

Posteriormente, una vez se haya traducido la consulta a lenguaje SQL, se crea un objeto el cual contendrá el objeto conexión y la consulta, y se mandará a PostgreSQL. Este proceso permitirá no introducir la conexión incorrectamente y darse cuenta una vez hayamos introducido la consulta.

-Introducción de consulta en álgebra relacional: Se genera un problema para poder introducir una consulta de álgebra relacional mediante un teclado tradicional, ya que la mayoría de operadores que intervienen no cuentan con una representación en el mismo. Por ello se ha decidido introducir un sistema de botones en la interfaz en el cual cada uno represente los distintos símbolos que se pueden utilizar en el álgebra relacional.

Para la representación de los símbolos mediante el lenguaje de programación Java se ha hecho uso de los caracteres Unicode. En dicha tabla de caracteres aparecen todos los operadores necesitados, por lo que la interfaz podrá mostrarlos y así crear una experiencia más intuitiva y completa.

-Transformación de consultas en álgebra relacional a consultas SQL: Este es el cuerpo del proyecto, es el aspecto más importante y el más difícil de desarrollar. A lo largo del proceso de desarrollo se han hecho distintos cambios para cambiar el enfoque a la hora de afrontar este aspecto. En esta parte explicaré por encima la forma en la que he tratado este problema, siendo más adelante cuando desarrolle con mayor exactitud la forma en la que funciona la aplicación y trata las distintas consultas.

Debido a la complejidad del lenguaje en álgebra relacional y la dificultad que aparece al transformar este a lenguaje SQL, se descartó desde el principio traducir 'carácter a carácter', ya que nunca llegaría a funcionar correctamente. El principal problema aparecería al encontrarnos con consultas complejas o subconsultas, ya que deben tratarse de forma distinta y realizando distintas operaciones. En estos casos se deben tratar desde la subconsulta más interna a la más externa e ir enlazando los resultados.

Por este motivo se decidió generar dos estructuras, una de paréntesis y otra de corchetes, la cual se construye una vez que el usuario haya introducido la consulta. En estas estructuras indicaremos cada par de paréntesis y corchetes mostrando su posición relativa dentro de los demás operadores iguales que él y su posición absoluta dentro de la consulta introducida. Gracias a esto se podrá tratar las subconsultas desde la más interna a la más externa y tratar también las condiciones de cada operador, las cuales deberán aparecer entre corchetes.

-Devolución de resultados: El último paso que se debe cumplimentar es el de mostrar los resultados que el SGBD genera. Si estuviéramos trabajando con alguna herramienta visual de representación como puede ser pgAdmin III, contaríamos con una herramienta donde se exponen dichos resultados. En nuestro caso debemos de crear un sistema de tablas para poder mostrar los resultados de las consultas introducidas.

Hablando en términos de programación se ha creado una clase llamada *CRUD*, la cual es la responsable de mostrar los resultados mediante la aplicación. La clase es una clase por defecto, usada en otros proyectos con el mismo fin, la cual ha sido modificada para adaptarla a las necesidades de esta aplicación. Mediante esta clase se podrán representar todos los resultados posibles que se puedan generar dependiendo de la sentencia introducida por el usuario.

Diseño

Diagrama de flujo

En este apartado se detallará el diagrama de flujo de la aplicación, para a continuación explicar el camino o caminos posibles que puede tomar. Lo primero es explicar los símbolos utilizados en el diagrama.

Inicio/Fin

Proceso

Entrada de Datos

Líneas de flujo

Display, Mostrar Datos

Existen más símbolos, pero en este caso solo he mostrado los que se han utilizado en esta aplicación. El diagrama de flujo quedaría de la siguiente forma:

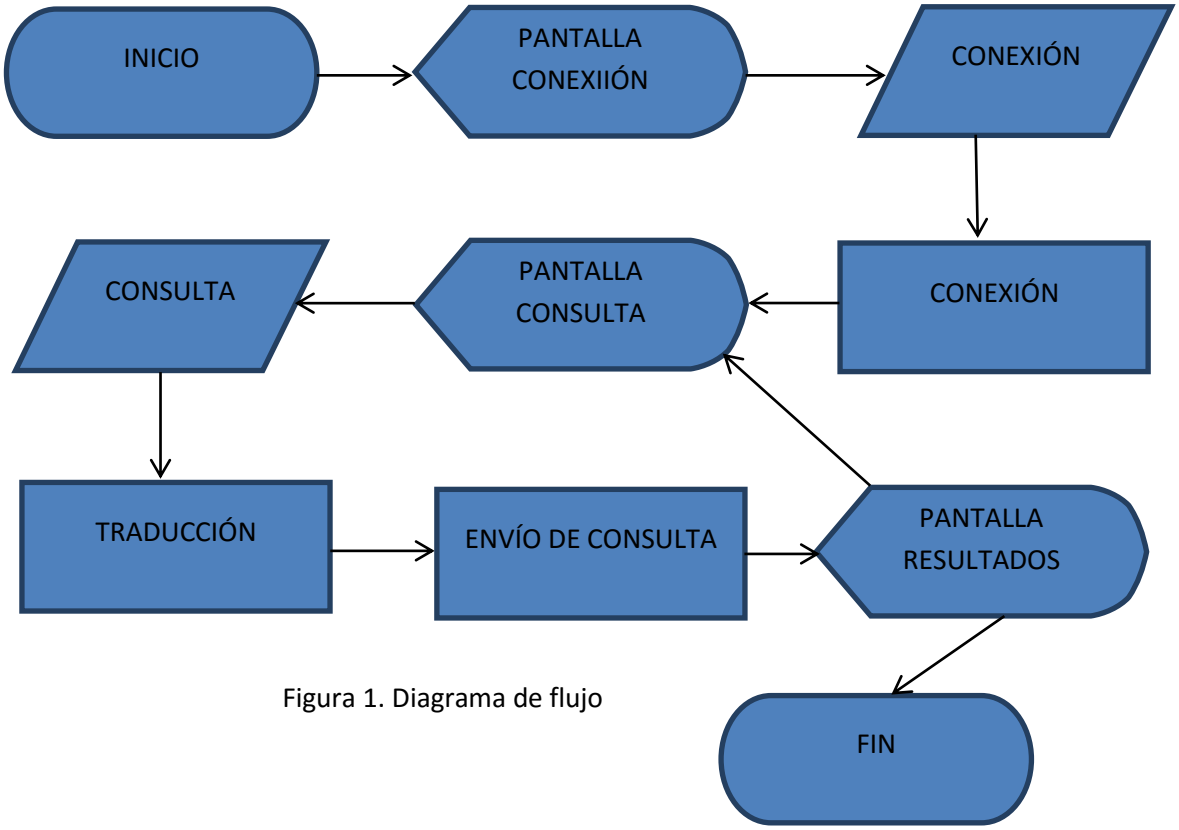


Figura 1. Diagrama de flujo

Este es el diagrama de flujo que toma la herramienta. A continuación se explicará internamente que ocurre en cada uno de los elementos haciendo el recorrido interno que realiza la aplicación paso a paso.

- Inicio: Se lanza la aplicación.

- Pantalla Conexión: Primera pantalla que aparece una vez se inicia la aplicación. La clase en Netbeans se llama Inicio.java. Esta pantalla está ideada para introducir los datos necesarios para poder realizar la conexión con la base de datos. Para ello hay espacio para introducir el nombre de la base de datos, el usuario, la contraseña, el puerto y la dirección IP. Una vez que se hayan introducido se debe de pulsar el botón de 'Conectar'.

Esta es la pantalla de Inicio:

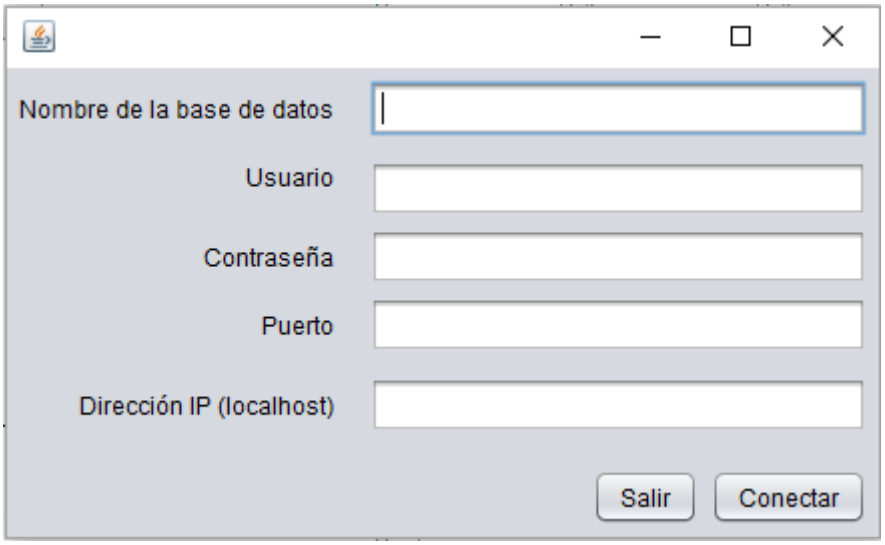


Figura 2. Pantalla Inicio

- Conexión (entrada de datos): Este paso corresponde al proceso en el cual el usuario introduce los datos comentados anteriormente en la pantalla que se muestra arriba para poder realizar la conexión.

- Conexión (proceso): A partir de los datos introducidos en el paso anterior, se tratan y se crea un objeto Conexión, el cual se envía al SGBD para crear la conexión. Este objeto solo contiene los datos de la conexión. La clase donde se desarrolla este proceso es Conexión.java.

- Pantalla Consulta: Pantalla cuya función es la de que el usuario pueda introducir la consulta que quiere que sea tratada. En Netbeans esta clase se llama Consulta.java. Cuenta con una serie de botones que representan símbolos matemáticos de jerarquía como pueden ser paréntesis o corchetes y los operadores del álgebra relacional.

Esta es la pantalla de Consulta:

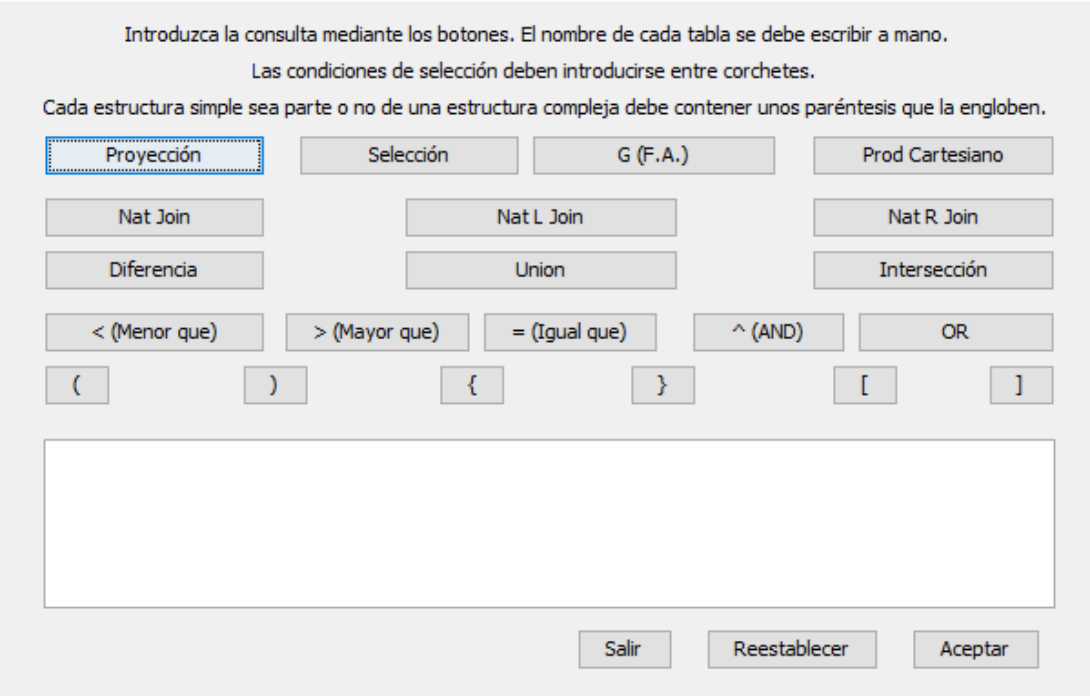


Figura 3. Pantalla Consulta

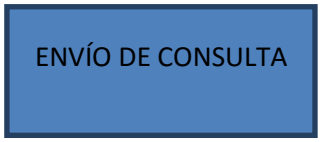
- Consulta (entrada de datos): Este paso se corresponde a la inserción de la consulta mediante los medios disponibles. El usuario podrá reestablecer la cadena introducida hasta el momento haciendo uso del botón 'Reestablecer' y una vez esté conforme con la consulta escrita deberá pulsar en 'Aceptar'.

- Traducción (proceso): Proceso más extenso de la aplicación, el cual se desarrolla dentro de la clase EstructuraConsulta.java, dentro del paquete 'Traductor'. Mediante este proceso se traduce la consulta introducida en el lenguaje del Álgebra Relacional a una consulta equivalente en lenguaje SQL.
  La clase se inicia al ser llamada por la clase Consulta.java. Comienza a recorrerse el constructor principal, según el cual se inicializan las variables globales y se crean las estructuras de paréntesis y corchetes de la consulta. Posteriormente llegamos a un punto importante, ya que es donde se comprueba si la consulta que ha sido introducida la asumimos como una sola consulta o varias de ellas unidas por símbolos del estilo al de Union (suma), Except (resta) o Intersect (intersección). Es importante

porque si la consideramos como varias deberá realizar recursividad para tratar cada una de ellas independientes a las demás. Siendo de cualquiera de las formas se llamará al método *transformar* para comenzar la traducción.

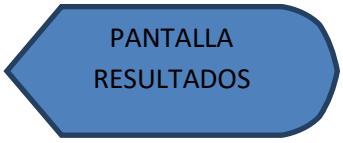
En este método se recorre la consulta, apoyándonos en la estructura de paréntesis, yendo desde las frases más internas a las más externas. Cada una de ella será analizada y dependiendo de que operador se encuentre generará una cadena u otra, con los datos que encuentre dentro de esa frase concreta. Esta cadena representará una consulta equivalente a lo encontrado en la frase que esté dentro de los paréntesis que se estén tratando. La cadena que se genere llevará una bandera del tipo __%tempX en algún lado donde posteriormente se sustituirá por otra cadena generada al analizar otra frase de la consulta. Cada cadena generada llevará también un indicador único del tipo id__X para poder renombrar cada frase de una forma única.

Las cadenas generadas se almacenan en un ArrayList *temporales*. Por último se ha creado un algoritmo por el cual se irán combinando todas las cadenas, a partir de la bandera __%tempX, hasta que quede una cadena final que será la consulta en lenguaje SQL. El último paso sería añadir un punto y coma al final de la consulta y cambiar los operadores del álgebra relacional por su correspondencia en SQL si hubiese tenido que haber recursividad.



- Envío de consulta (proceso): Proceso según el cual se envía la consulta ya traducida al SGBD. Para ello se crea un objeto *CRUD* el cual está compuesto por la consulta y por un objeto conexión, el cual se ha creado anteriormente. Aunque ya hayamos realizado una conexión anteriormente, es necesario volver a realizarla cada vez que queramos enviar una nueva consulta.

  La clase *CRUD.java* es la encargada de enviar los datos y también de recibir los resultados. En esta clase se organizan los resultados mediante tablas y columnas para facilitar el visualizado al usuario.



- Pantalla de resultados: Pantalla cuya función es la de mostrar los resultados que devuelve el SGBD a partir de la consulta que le ha sido enviada. Esta pantalla cuenta con un botón 'Hacer otra consulta' según el cual volveríamos a la pantalla de Consulta, como se puede ver en el diagrama de flujo.
-

- Fin: Se cierra la aplicación.

Arquitectura e implementación

En este apartado se explicará la arquitectura con la que cuenta la herramienta. Lo dividiré por paquetes y clases y para cada una de ellas se comentarán sus estructuras y sus métodos principales. Durante este apartado no se desarrollará la lógica de la aplicación ya que lo haré en otro apartado.

Interfaces Gráficas

El primer paquete que se explicará será el de Interfaces Gráficas en él se encuentran las clases responsables de aportar una interfaz para que el usuario pueda recorrer y hacer uso de la aplicación. Las clases que la componen son: Inicio, Consulta y GUI.

Figura 4. Diagrama de clases Interfaces Gráficas

Inicio

Clase responsable de mostrar la primera pantalla que aparece al ejecutar la herramienta. Esta clase cuenta con cinco cuadros de texto diseñados para que el usuario pueda introducir todos los datos necesarios para la conexión con la base de datos. Cuenta también con dos botones, uno para cerrar la aplicación '*Salir*' y otro para realizar la conexión una vez que se hayan introducido todos los datos '*Conectar*' y avanzar a la siguiente pantalla en la que se introducirá la consulta.

Al ser la pantalla de inicio de nuestra aplicación, cuenta con un método '*main*' que indica al compilador dónde debe de empezar y cuál es la primera ventana que debe de mostrar. Debido a que es una clase diseñada solo para el aspecto gráfico, esta clase no cuenta con ningún método ni estructura especial la cual deba de ser comentada.

Consulta

Clase responsable de mostrar la pantalla por la cual el usuario deberá introducir la consulta deseada. Para ello cuenta con una serie de botones que representan cada uno de los símbolos del Álgebra Relacional así como operadores matemáticos necesarios para poder completar la consulta correctamente. Al igual que la anterior clase de '*Inicio*' cuenta con varios botones para la navegación: '*Salir'* para cerrar la aplicación, '*Reestablecer'* para limpiar el cuadro donde irá apareciendo la consulta según el usuario la vaya introduciendo y '*Aceptar*' para avanzar a la siguiente venta y enviar la consulta para su posterior traducción.

Los operadores del Álgebra Relacional son símbolos los cuales no se encuentra en un teclado tradicional, para ello se ha hecho uso del alfabeto Unicode para poder representar dichos símbolos en la herramienta. Debido a la misma razón que la clase de '*Inicio',* esta clase no cuenta con ningún método ni estructura especial que deba de ser comentada.

GUI

Clase responsable de mostrar por pantalla los resultados que devuelve el PostgreSQL a la consulta lanzada anteriormente. Es una clase por defecto que se añade al configurar la conexión con la base de datos. A esta clase por defecto se le ha añadido un botón que aparece como '*Hacer otra búsqueda*' mediante el cual podremos volver a la pantalla de '*Consulta*' para poder realizar otra consulta sobre la misma conexión.

Esta clase cuenta con un método llamado '*LoadData()'* el cual se ejecuta una vez se pulse sobre el botón '*Hacer otra consulta*'. La herramienta vuelve a lanzar la ventana de la consulta manteniendo los datos de conexión ya introducidos. Esta acción puede repetirse tantas veces como uno quiera.

Conexión

Este paquete contiene las clases responsables para la conexión de la aplicación con la base de datos. Se explicarán sus principales métodos y estructuras así como su comunicación con las distintas clases. Las clases que conforman este paquete son: CRUD y Conexión.

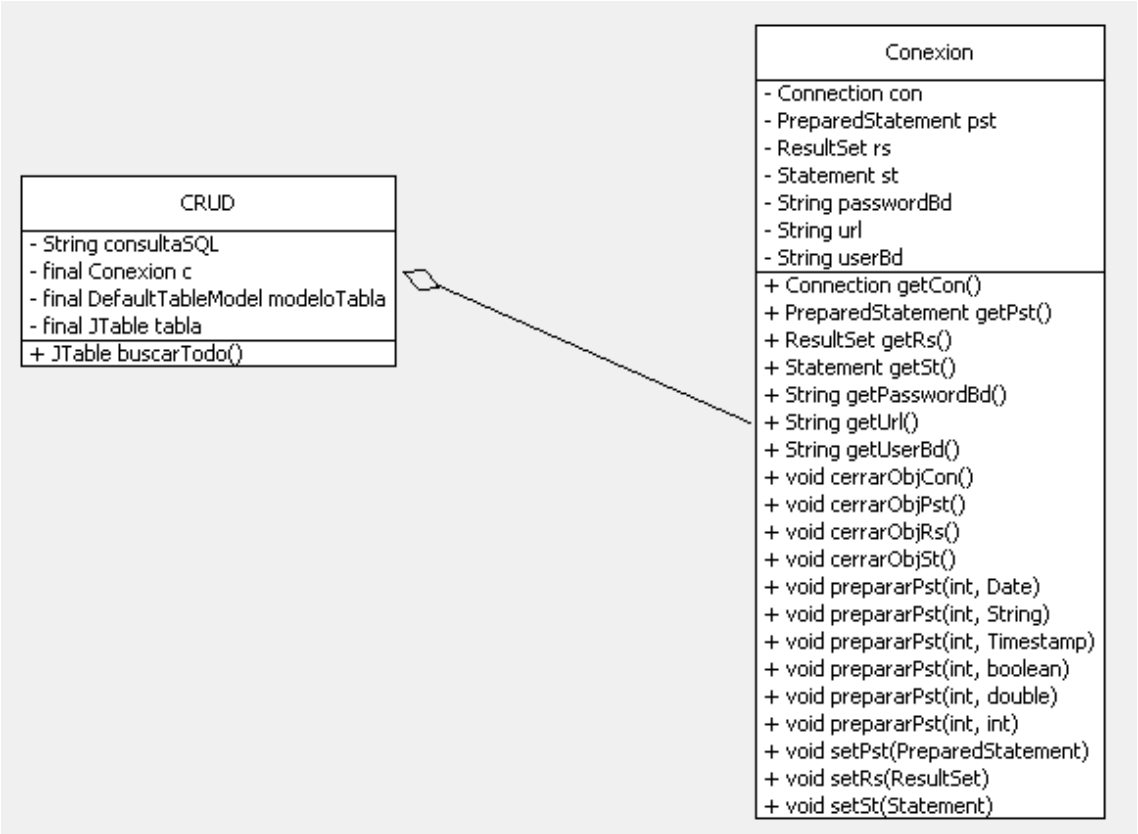


Figura 5. Diagrama de clases Conexion

CRUD

Clase responsable de enviar nuestra consulta ya traducida a la base de datos, recoger los resultados que se devuelve y organizarlos mediante una tabla de filas y columnas para poder ser visualizados. Al igual que la clase '*GUI*' es una clase por defecto sobre la cual se han ido añadiendo instrucciones para adecuarla a las necesidades.

Como ya se ha dicho anteriormente, cuenta con una serie de instrucciones para recoger los resultados de la búsqueda y organizarlos en una estructura de filas y columnas para que el usuario pueda ver el resultado de su búsqueda. Aparte de estas instrucciones, y al ser una clase por defecto, no cuenta con ninguna estructura o método especial que deba de ser comentado.

Conexión

Clase responsable de generar la estructura de la conexión. Los datos de la misma los recibe provenientes de la clase '*Inicio*', se encarga de organizarlos de forma que el Sistema de Gestión de Bases de Datos (SGBD) pueda reconocerlos y los envía a la clase '*CRUD*'. Para ello se crea un objeto *Conexión* con todos los datos necesarios.

Esta clase cuenta con métodos *get* y *set* para los distintos atributos que forman parte la estructura así como los métodos para cerrar los distintos objetos de conexión (*con, st, pst* y *rs).*

Traductor

Este paquete solo contiene la clase *'EstructuraConsulta'*, la cual se encarga de traducir la consulta del Álgebra Relacional al lenguaje SQL.

Figura 6. Diagrama de clases Traductor

EstructuraConsulta

Esta clase es la encargada de traducir la consulta, por lo que es la clase que contiene la mayor parte de la lógica de la herramienta. Cuenta con una cantidad importante de métodos y de estructuras por lo que se hablará de ellas individualmente comentando su función dentro de la aplicación. Se explicarán también sus constructores y cuando se utiliza cada uno de ellos.

En primer lugar se tratarán las distintas estructuras más importantes que se han generado:

- Contadores: Variables del tipo Entero las cuales se utilizan a lo largo de la clase para llevar distintas cuentas. Estas cuentas pueden ser de paréntesis, corchetes o de caracteres, según sea la necesidad. La mayoría de su uso se ubica en el método más importante, *transformar*, el cual se detallará más adelante.

- int[][] parentesis: Matriz de tipo entero utilizada para representar la ubicación de todos los paréntesis del texto introducido. La primera posición representa el orden que ocupa el paréntesis dentro de la frase. Por ejemplo, el primer par de paréntesis que aparezca en la frase tendrá la posición cero ([0][0], [0][1]), el siguiente la uno

([1][0], [1][1]) y así sucesivamente. La segunda posición indicará si se habla del paréntesis de apertura o de clausura. En el caso del paréntesis de apertura aparecerá un 0 y en el de clausura un 1. Por lo tanto cada valor de la primera posición deberá estar asociado con los dos valores posibles de la segunda posición, como se puede ver en el ejemplo previamente mostrado.
El valor asignado para cada combinación será la posición que ocupa ese paréntesis en la frase. Un pequeño ejemplo con una frase sencilla sería así:

((a)) -> [0][0]=0; [0][1]=4; [1][0]=1; [1][1]=3;

Esta estructura nos será muy útil para tratar las subconsultas de una forma jerárquica.

- int[][] corchetes: Matriz de tipo entero utilizada para representar la ubicación de todos los corchetes del texto introducido. El funcionamiento y la estructura es exactamente igual que la matriz de paréntesis por lo que no se volverá a desarrollar lo mismo.

- ArrayList<OperacionFrase> opFrases: ArrayList de *OperacionFrase*, el cual es un objeto formado por tres variables: *operacion*, *posicion* y *esPrincipal*. *Operacion* es un char que contiene el operador del álgebra relacional en cuestión. *Posicion* es un entero que representa la posición de dicha operación dentro de la frase. *EsPrincipal* es un booleano que representa si esa operación forma parte de una frase principal o no. El objetivo de este ArrayList es ayudar a determinar si se debe realizar recursividad o no sobre una cadena concreta.

- ArrayList<Character> operaciones: ArrayList de caracteres que indican cuales son las operaciones que van a necesitar recursividad sobre su transformación. Estas operaciones serán las de Unión, Intersección y Diferencia.

- ArrayList<String> temporales: ArrayList de String en el cual se irán almacenando las frases temporales, que representan consultas SQL internas. Estos temporales son traducciones parciales de la frase introducida la cual va transformándose gradualmente según un orden jerárquico. Estas frases se irán almacenando según un orden preciso en este ArrayList para después irse combinando hasta resultar la frase traducida completa.

Una vez definidas las principales estructuras se detallarán los dos constructores que aparecen en esta clase y la función de cada uno de ellos:

- public EstructuraConsulta(String s, String token, int id): Es el constructor principal. Este constructor recibe tres parámetros: s es un String que representa la consulta introducida por el usuario y token e id son unas variables que se utilizan posteriormente combinados para renombrar las subconsultas con un valor único.
  En este constructor se realizan las llamadas para inicializar las variables, crear las distintas estructuras, realizar si fuese necesaria la recursividad y llama al método *transformar* encargado de realizar la traducción de la consulta.

- public EstructuraConsulta(String s): Constructor alternativo que recibe un solo parámetro: s es un String que representa una frase que hayamos querido enviar al mismo. Este constructor lo único que hace es crear una estructura de corchetes sobre el String que hayamos enviado. Para ello inicializa las variables globales correspondientes, inicializa los corchetes de esa frase y crea una estructura de ellos, independiente de la creada sobre la consulta total.
  La función de este constructor es ayudar en la traducción al encontrar una función de agregación (g) ya que debido a su estructura no se puede traducir al igual que otras funciones del álgebra relacional. Este aspecto se tratará con mayor detalle posteriormente.

Por último queda hablar de los distintos métodos que forman parte de esta clase, los cuales se detalla a continuación en el orden que aparecen en el código:

- private OperacionFrase hayPrincipales(): Método privado el cual recorre cada frase de la estructura *opFrases* y comprueba si el booleano está en true y por lo tanto la frase es principal. Este método es llamado por el constructor *EstructuraConsulta*.

- private int numParentesisAbiertos(String cadena): Método privado el cual recibe un String, que será la consulta, y devuelve un entero el cual representa el número de paréntesis abiertos que contiene, o lo que es lo mismo, el número de pares de paréntesis . Este proceso es útil para conocer con cuantos tratamos para poder rellenar luego su matriz correspondiente. Este método es llamado por los métodos *transformar* e *InicializarVariablesGlobales.*

- private int numCorchetes(): Método privado que toma la cadena que representa la consulta introducida y devuelve un entero el cual representa el número de pares de corchetes que se encuentran en ella. Es un método similar al de *numParentesisAbiertos.* Este método es llamado por el método *InicializarVariablesGlobales.*

- private void crearEstructuraPar(): Método privado que usa la cadena de la consulta introducida y crea la estructura de paréntesis *int[][] parentesis*, aparte de esto va rellenando la estructura *ArrayList<OperacionFrase> opFrases* a medida que va recorriendo la cadena. El método cuenta con un bucle for el cual va recorriendo el String y buscando paréntesis de apertura, de clausura y operaciones. Cuenta con varios contadores para saber el orden de los paréntesis y poder crear la estructura correctamente. Este método es llamado por el constructor *EstructuraConsulta*.

- private void crearEstructuraCor(): Método privado que usa la cadena de la consulta introducida y crea la estructura de paréntesis *int[][] corchetes*. Su funcionamiento es similar a *crearEstructuraPar()* con la diferencia que este método está orientado a corchetes y aparte no rellena el ArrayList *opFrases* ya que este cometido lo tiene el anterior método. Este método puede ser llamado por los dos constructores *EstructuraConsulta.*

- private void transformar(): Método privado que usa la cadena introducida para transformarla al lenguaje SQL. Se apoya en la estructura de paréntesis para ir desgranando la consulta de dentro hacia fuera. Cada subcadena que recoge la analiza y dependiendo del símbolo que encuentre dentro de ella realizará unas acciones u otras. Cada resultado de traducir cada una de las subcadenas se almacenan en el ArrayList de *temporales* con un identificador único ('__%tempX'). A partir de este identificador único se irán combinando las distintas subcadenas para generar una cadena final. Para esto último se ha creado un bucle *for* mediante el cual se irá intercambiando cada identificador único por el temporal correspondiente. Este método es llamado por el constructor *EstructuraConsulta.*

- private int ultimoIndiceParAbierto(): Método privado que devuelve un entero. El método es utilizado por el método *crearEstructuraPar* para determinar cuál es el último paréntesis abierto que se ha metido en el ArrayList de *paréntesis* y no ha sido cerrado todavía. Para ello comprueba que posición del ArrayList tiene como primera posición un valor distinto a -1 y un valor en la segunda posición de -1. Esto funciona porque previamente se ha rellenado el ArrayList con valores -1 para todos los casos. El entero que devuelve este método corresponde a la posición del paréntesis comentado anteriormente.

- private int ultimoIndiceCorAbierto(): Método privado con un funcionamiento y objetivo similar a *ultimoIndiceParAbierto()*. En este caso queremos conocer la posición del último corchete abierto que se ha metido en el ArrayList de *corchetes* y no ha sido cerrado todavía. Este método es llamado por el método *CrearEstructuraCor.*

- private void inicializarParentesis(): Método privado mediante el cual se rellena el ArrayList de *paréntesis* con valores todos ellos de -1. Esto será útil para el método *ultimoIndiceParAbierto()* para conocer aquellos valores que han sido modificados y cuáles no. Este método es llamado por el constructor *EstructuraConsulta.*

- *private void inicializarCorchetes()*: Método privado mediante el cual se rellena el ArrayList de *corchetes* con valores todos ellos de -1. Tiene un funcionamiento similar al método *inicializarParentesis()*. Esto será útil para el método *ultimoIndiceCorAbierto()* para conocer aquellos valores que han sido modificados y cuáles no. Este método puede ser llamado por ambos constructores *EstructuraConsulta.*

- private void InicializarVariablesGlobales(): Método privado cuyo cometido es inicializar las variables globales. Inicializa las variables globales que se utilizan a lo largo de la clase y las distintas estructuras que se han comentado al comienzo del capítulo (*paréntesis, corchetes, opFrases, temporales* y *operaciones*). Este método puede ser llamado por los dos constructores *EstructuraConsulta.*

- private boolean estaEnFrasePrincipal(OperacionFrase operacionPrincipal): Método privado que recibe un objeto del tipo OperacionFrase y devuelve un boolean. El método sirve para decir si la cadena con la que estamos tratando es parte de una consulta o es una consulta en sí misma. Es utilizado por el constructor *EstructuraConsulta* para determinar si es necesario utilizar recursividad o no.

- public String getCadenaFinal(): Método público que devuelve un String. El método toma la consulta ya traducida y realiza las últimas modificaciones antes de ser enviada al SGBD. En el caso en el que se haya realizado la recursividad sustituye los operadores restantes que no hayan sido traducidos por su traducción al SQL y aparte de esto añade ';' a la cadena para que pueda ser entendida por el sistema. Este método es llamado desde la clase *Consulta*.

- private class OperacionFrase: Clase privada la cual define la estructura del objeto *OperacionFrase*, el cual ha sido definido previamente.

Testing

A continuación se realizarán las pruebas de testeo en busca de errores en la ejecución del código. Se tratarán de buscar combinaciones para comprobar que la aplicación funciona correctamente ante todos los posibles casos de uso. Para ello lo primero que se hará será crear una base de datos nueva sobre la que se ejecutarán las consultas.

En primer lugar se debe descargar el software Toad Data Modeler para generar la estructura de nuestra base de datos para después crear dicha base de datos mediante pgAdminIII. Lo primero será generar el modelador que sea compatible con SGBD. Para ello seleccionamos la opción que sea necesaria, en nuestro caso PostgreSQL 9.4, y pulsamos sobre 'yes' en el mensaje emergente.

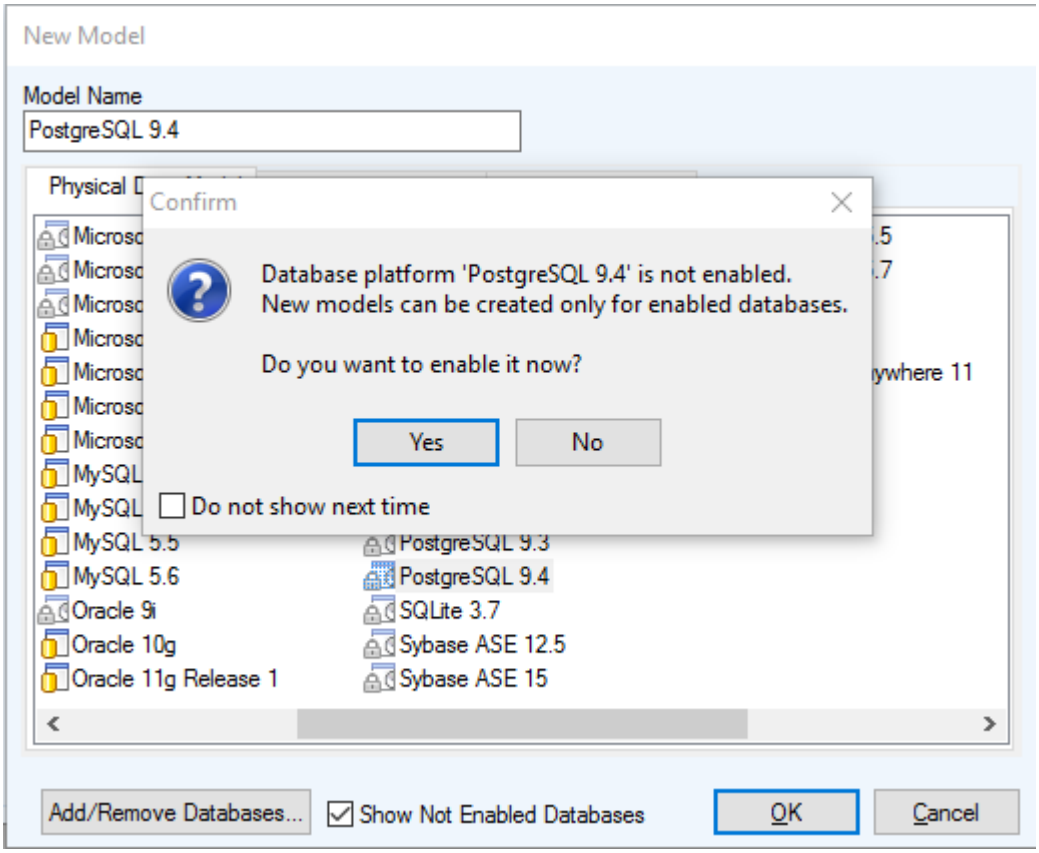


Figura 7. Generar modelador

Una vez hecho esto, se procede a crear la estructura. Para ello se van añadiendo cada entidad con sus atributos y las relaciones que existen entre ellos. En este caso, y para darle una mayor realidad, se ha creado una base de datos simple orientada a la que podría tener un colegio, donde se pueden almacenar datos de alumnos, profesores y departamentos. Se ha creado una relación de N:M entre alumnos y profesores, ya que un alumno puede tener varios profesores y un profesor puede impartir clase a varios alumnos. Por otro lado se ha creado una

relación de 1:N entre profesor y Departamento ya que cada profesor debe pertenecer a un solo departamento y a un departamento pueden pertenecer varios profesores.

El esquema queda de la siguiente forma:

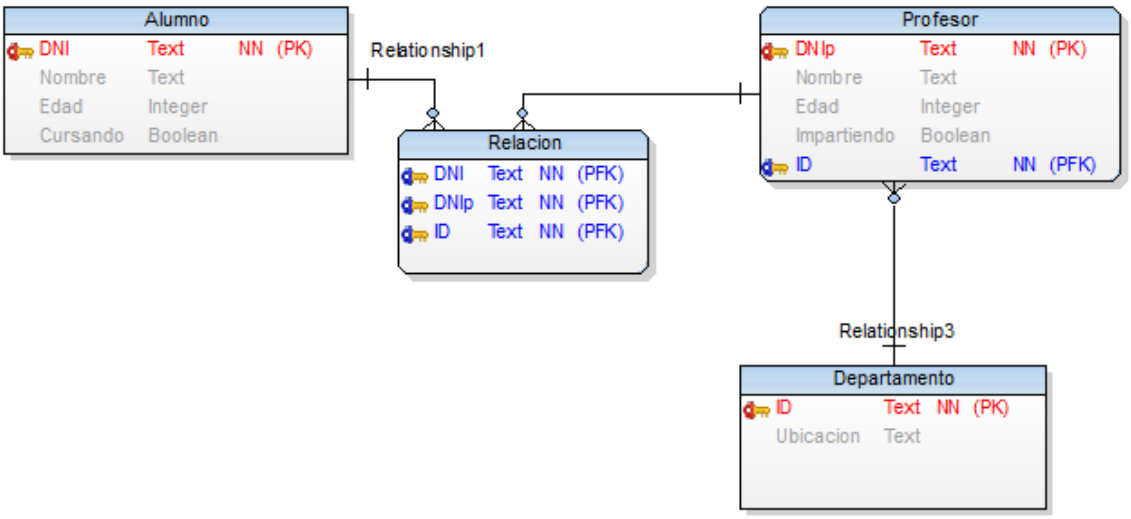


Figura 8. Esquema BBDD TESTING

El siguiente paso es generar el script el cual se va a cargar en la base de datos. Para generarlo se va a las pestañas de la parte superior y haremos Model -> Generate DDL Script -> Run… Aparece una ventana emergente y se pulsa sobre 'Generate'.

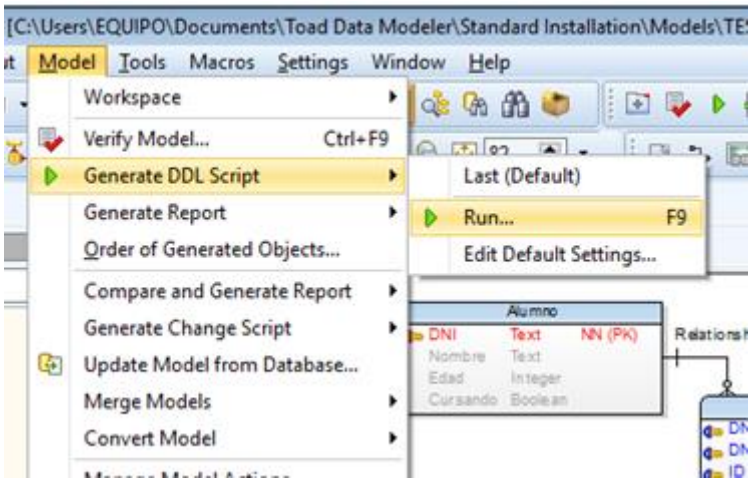
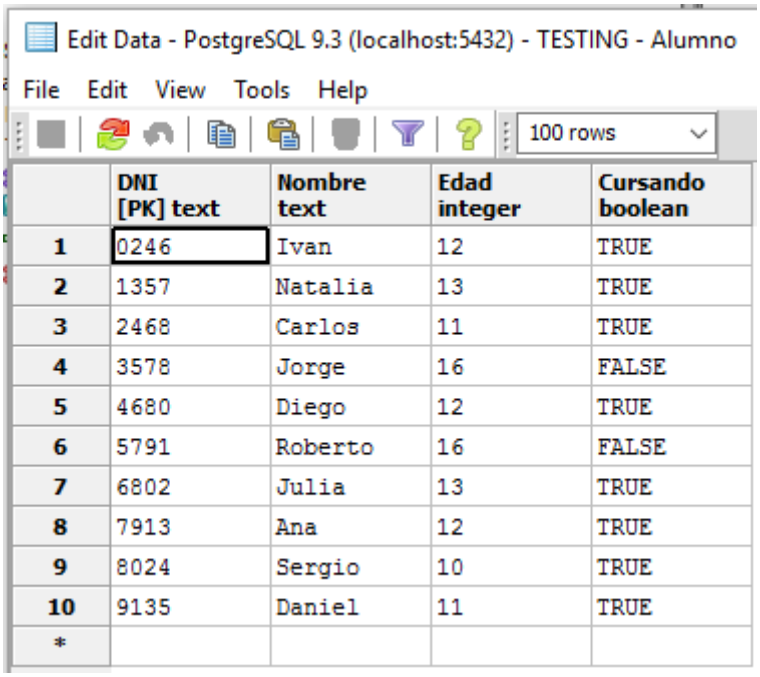


Figura 9. Generar Script TESTING

Una vez generado el script, se debe de crear una nueva base de datos local mediante el software pgAdmin III. En este caso la llamaremos 'TESTING'. Sobre esta base de datos se carga el Script generado anteriormente y ya está nuestra base de datos casi lista para poder realizar el testeo que se quiera. El último paso que falta es el de cargar datos, para ello habrá

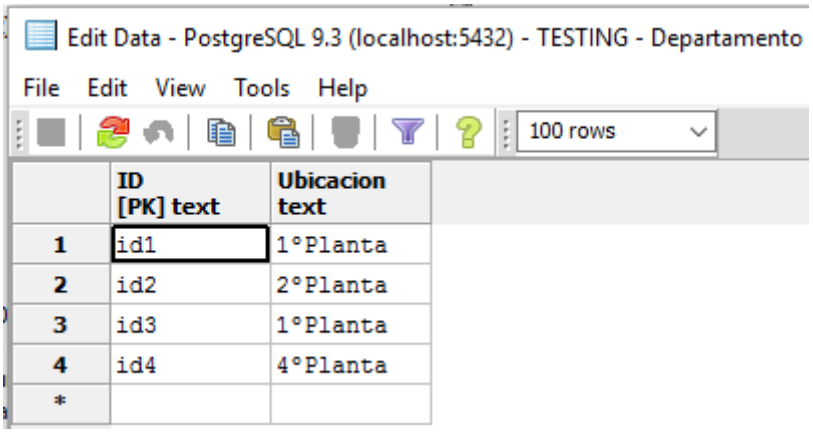
que irlos introduciendo manualmente teniendo en cuenta las restricciones que existen debido a las relaciones que se han generado entre nuestras entidades.

A continuación muestro como quedaría cada tabla después de la inserción para poder comprobar después si las consultas se realizan correctamente.



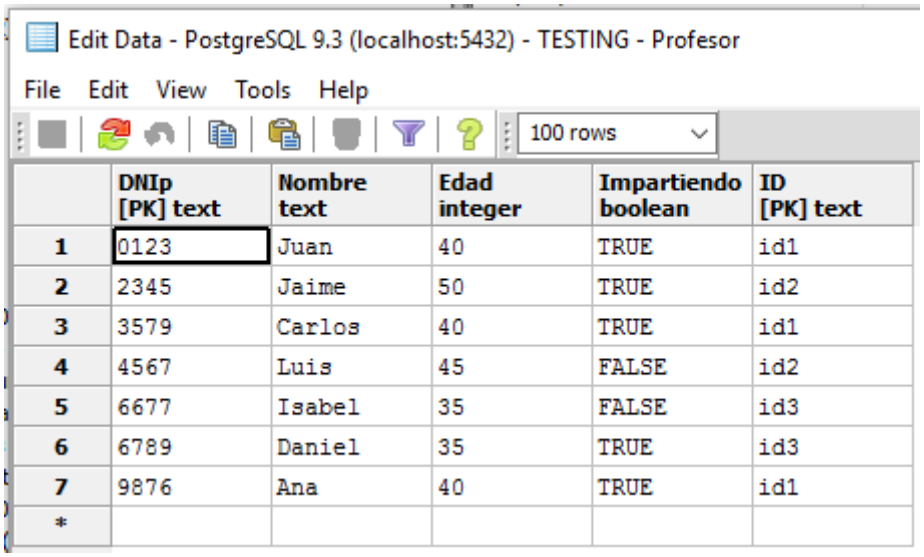
	DNI [PK] text	Nombre text	Edad integer	Cursando boolean
1	0246	Ivan	12	TRUE
2	1357	Natalia	13	TRUE
3	2468	Carlos	11	TRUE
4	3578	Jorge	16	FALSE
5	4680	Diego	12	TRUE
6	5791	Roberto	16	FALSE
7	6802	Julia	13	TRUE
8	7913	Ana	12	TRUE
9	8024	Sergio	10	TRUE
10	9135	Daniel	11	TRUE
*				

Figura 10. Datos Alumno



	ID [PK] text	Ubicacion text
1	id1	1°Planta
2	id2	2°Planta
3	id3	1°Planta
4	id4	4°Planta
*		

Figura 11. Datos Departamento



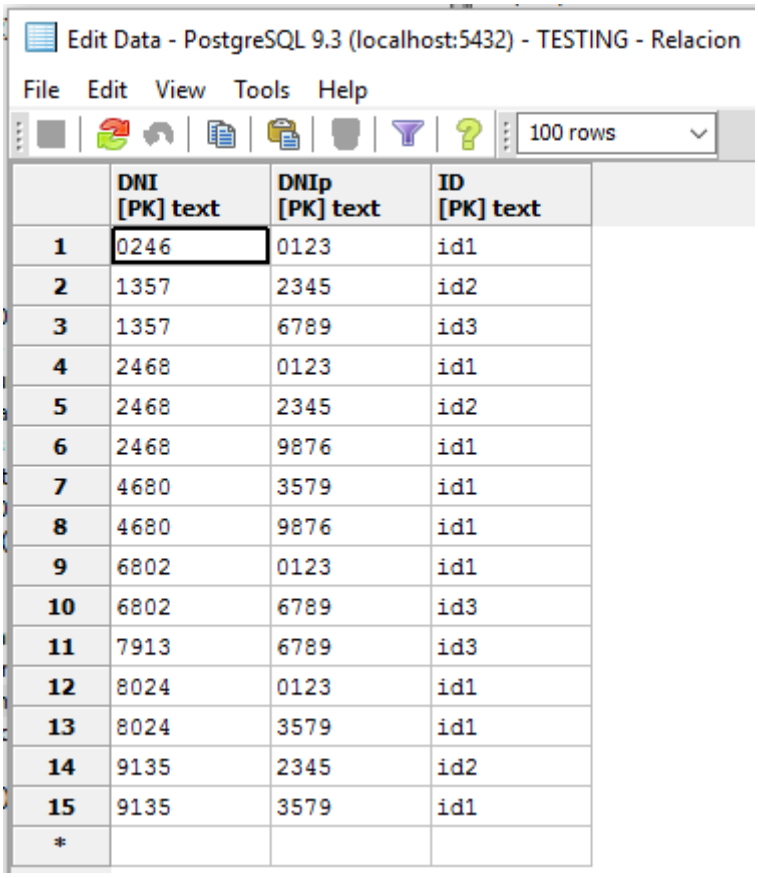
Edit Data - PostgreSQL 9.3 (localhost:5432) - TESTING - Profesor

File Edit View Tools Help

100 rows

	DNIp [PK] text	Nombre text	Edad integer	Impartiendo boolean	ID [PK] text
1	0123	Juan	40	TRUE	id1
2	2345	Jaime	50	TRUE	id2
3	3579	Carlos	40	TRUE	id1
4	4567	Luis	45	FALSE	id2
5	6677	Isabel	35	FALSE	id3
6	6789	Daniel	35	TRUE	id3
7	9876	Ana	40	TRUE	id1
*					

Figura 12. Datos Profesor



Edit Data - PostgreSQL 9.3 (localhost:5432) - TESTING - Relacion

File Edit View Tools Help

100 rows

	DNI [PK] text	DNIp [PK] text	ID [PK] text
1	0246	0123	id1
2	1357	2345	id2
3	1357	6789	id3
4	2468	0123	id1
5	2468	2345	id2
6	2468	9876	id1
7	4680	3579	id1
8	4680	9876	id1
9	6802	0123	id1
10	6802	6789	id3
11	7913	6789	id3
12	8024	0123	id1
13	8024	3579	id1
14	9135	2345	id2
15	9135	3579	id1
*			

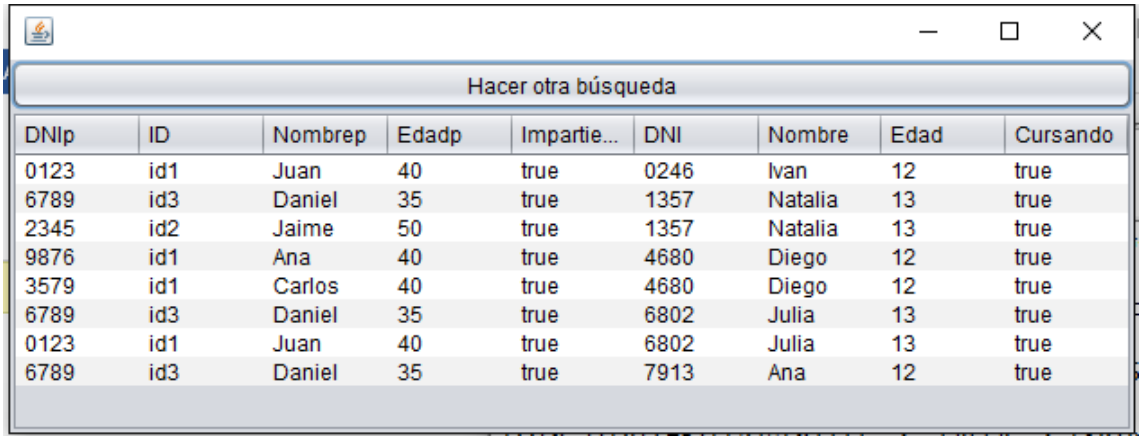
Figura 13. Datos Relacion

Una vez están los datos introducidos, se van a realizar varias consultas sobre esta base de datos comparando los resultados con los que deberían dar introduciendo la consulta equivalente en SQL directamente.

La primera consulta que se realizará nos dará un listado de todos los alumnos que tengan una edad superior a los 11 años y sus profesores asociados. La consulta que se insertará en Álgebra Relacional sería la siguiente:

(σ{"Edad">11}(("Profesor")⋈(("Relacion")⋈("Alumno"))))

Y el resultado que devuelve la herramienta es el siguiente:



DNIp	ID	Nombrep	Edadp	Impartie...	DNI	Nombre	Edad	Cursando
0123	id1	Juan	40	true	0246	Ivan	12	true
6789	id3	Daniel	35	true	1357	Natalia	13	true
2345	id2	Jaime	50	true	1357	Natalia	13	true
9876	id1	Ana	40	true	4680	Diego	12	true
3579	id1	Carlos	40	true	4680	Diego	12	true
6789	id3	Daniel	35	true	6802	Julia	13	true
0123	id1	Juan	40	true	6802	Julia	13	true
6789	id3	Daniel	35	true	7913	Ana	12	true

Figura 14. Resultados 1

Dicha consulta la traducimos al lenguaje SQL manualmente y quedaría de la siguiente forma:

Select * from ("Alumno" natural join "Relacion" natural join "Profesor") where "Edad">11;

Si ejecutamos esta consulta en PostgreSQL directamente nos devuelve los siguientes resultados:

	DNIp text	ID text	DNI text	Nombre text	Edad integer	Cursando boolean	Nombrep text	Edadp integer	Impartiendo boolean
1	0123	id1	0246	Ivan	12	t	Juan	40	t
2	6789	id3	1357	Natalia	13	t	Daniel	35	t
3	2345	id2	1357	Natalia	13	t	Jaime	50	t
4	9876	id1	4680	Diego	12	t	Ana	40	t
5	3579	id1	4680	Diego	12	t	Carlos	40	t
6	6789	id3	6802	Julia	13	t	Daniel	35	t
7	0123	id1	6802	Julia	13	t	Juan	40	t
8	6789	id3	7913	Ana	12	t	Daniel	35	t

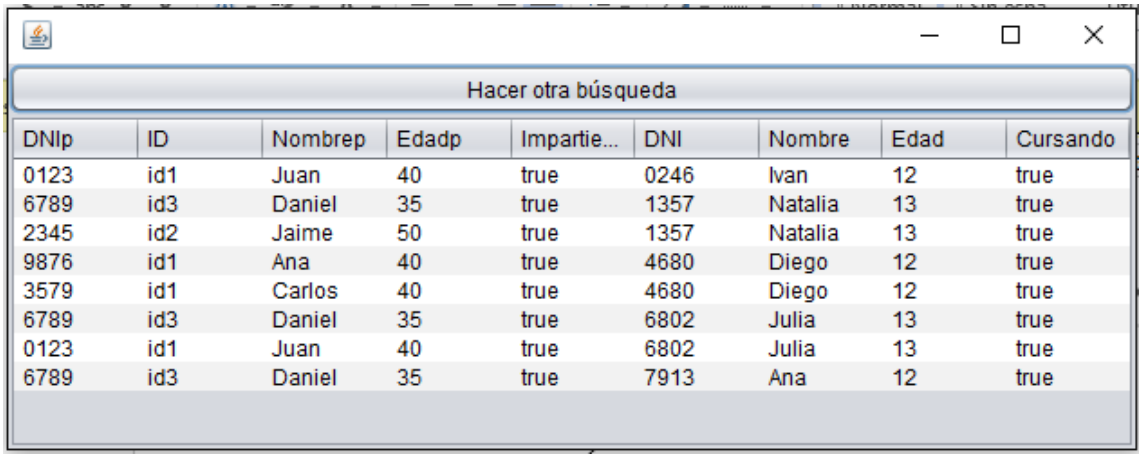
Figura 15. Resultados 2

Como se puede ver comparando ambas imágenes el resultado es el mismo, por lo que se puede decir que en esta prueba el programa funciona correctamente. Podemos realizar más pruebas sobre esta misma consulta. Para ello aplicamos las reglas de la heurística sobre las consultas y volvemos a hacer la misma prueba.

La consulta en Álgebra Relacional aplicando heurística quedaría de la siguiente forma:

(("Profesor")⋈(("Relacion")⋈(σ{"Edad">11})("Alumno")))

Los resultados de ejecutar esta sentencia son los siguientes:



DNIp	ID	Nombrep	Edadp	Impartie...	DNI	Nombre	Edad	Cursando
0123	id1	Juan	40	true	0246	Ivan	12	true
6789	id3	Daniel	35	true	1357	Natalia	13	true
2345	id2	Jaime	50	true	1357	Natalia	13	true
9876	id1	Ana	40	true	4680	Diego	12	true
3579	id1	Carlos	40	true	4680	Diego	12	true
6789	id3	Daniel	35	true	6802	Julia	13	true
0123	id1	Juan	40	true	6802	Julia	13	true
6789	id3	Daniel	35	true	7913	Ana	12	true

Figura 16. Resultados 3

Como puede comprobarse aparecen los mismos resultados por lo que para esta consulta sigue funcionando correctamente la aplicación.

Se va a realizar una prueba más con otro tipo de consulta, esta vez utilizando la Unión y haciendo un tipo de consulta por la cual la aplicación tiene que hacer recursividad para poder traducirla. La consulta lista el nombre y la edad de los alumnos que tengan 11 años y de los profesores que tengan 40 años.

La consulta que hay que temer en lenguaje de Álgebra relacional es la siguiente:

(Π{"Nombre","Edad"}(σ{"Edad"=11}("Alumno")))∪
(Π{"Nombrep","Edadp"}(σ{"Edadp"=40}("Profesor")))

Si se ejecuta esa consulta en la aplicación nos devuelve los siguientes resultados:

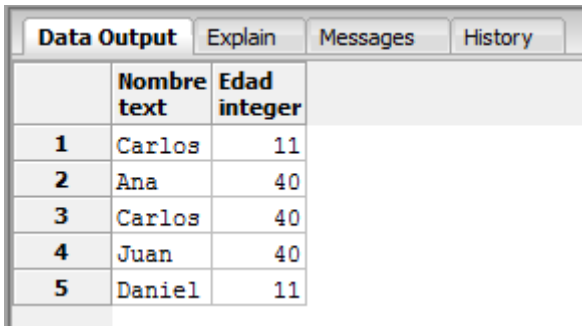
Nombre	Edad
Carlos	11
Ana	40
Carlos	40
Juan	40
Daniel	11

Figura 17. Resultados 4

El equivalente de esta consulta en SQL sería:

(Select "Nombre","Edad" from "Alumno" where "Edad"=11) Union (Select "Nombrep","Edadp" from "Profesor" where "Edadp"=40);

Si ejecutamos esta consulta mediante PostgreSQL nos dará el siguiente resultado:



	Nombre text	Edad integer
1	Carlos	11
2	Ana	40
3	Carlos	40
4	Juan	40
5	Daniel	11

Figura 18. Resultados 5

Como se puede ver en este otro ejemplo la aplicación funciona correctamente con distintos tipos de consulta. Con esto termino el apartado de testeo concluyendo que la herramienta funciona como debería funcionar.

Manual de usuario

A continuación voy a realizar un manual para ayudar al usuario a desenvolverse por la aplicación. Además voy a indicar varias de las reglas que hay que seguir para poder introducir las consultas correctamente para que la aplicación pueda entenderlas y traducirlas.

El primer paso, como es obvio, es ejecutar la aplicación dentro de nuestro ordenador. La primera pantalla que aparece es igual a la siguiente imagen:

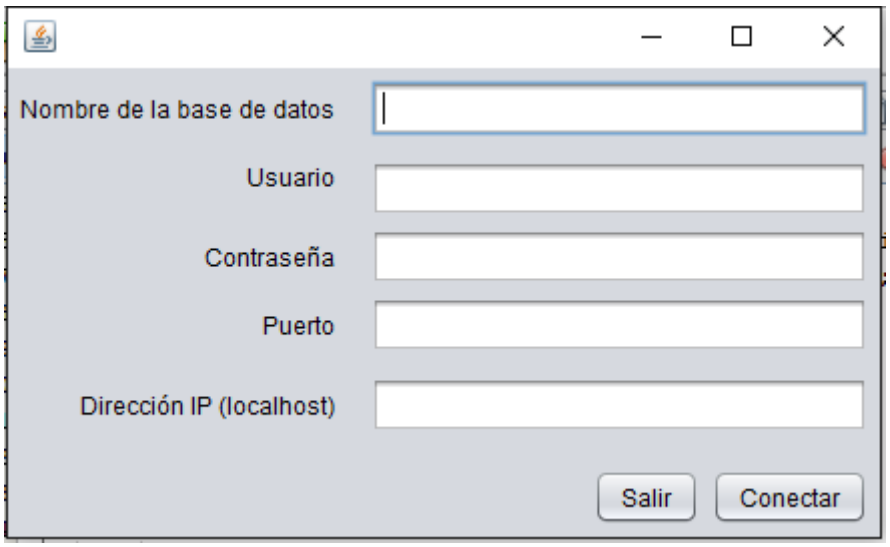


Figura 19. Pantalla Inicio

Como se puede ver es el momento en el que se deberán introducir los datos de la conexión. En primer lugar se introducirá el nombre la base de datos, este dato debe aparecer exactamente igual que como aparece en PostgreSQL, teniendo en cuenta espacios y letras en mayúsculas o minúsculas. Después se deberá introducir el usuario del SGBD con el que queremos acceder y su contraseña. El siguiente paso es introducir el puerto en el que se encuentra la base de datos. Para este caso, no es que sea una regla general, pero PostgreSQL suele otorgar por defecto el puerto '5432' para sus conexiones. Si este no es el caso basta con acceder a la BBDD mediante pgAdmin III y consultar el puerto de la base de datos. Por último debemos introducir la dirección IP. En el caso que se esté trabajando sobre una base de datos local, en nuestro ordenador, la dirección IP será 'localhost', en caso contrario se introducirá la dirección IP correspondiente.

Una vez se tengan todos los campos rellenos, pulsar el botón de conectar. Se deben de revisar los datos una vez se hayan introducido antes de pulsar el botón. Para evitar pérdidas de tiempo innecesarias se recomienda que se revisen estos datos para facilitar el funcionamiento.

Este es un ejemplo de una conexión a una base de datos llama *Prueba*:

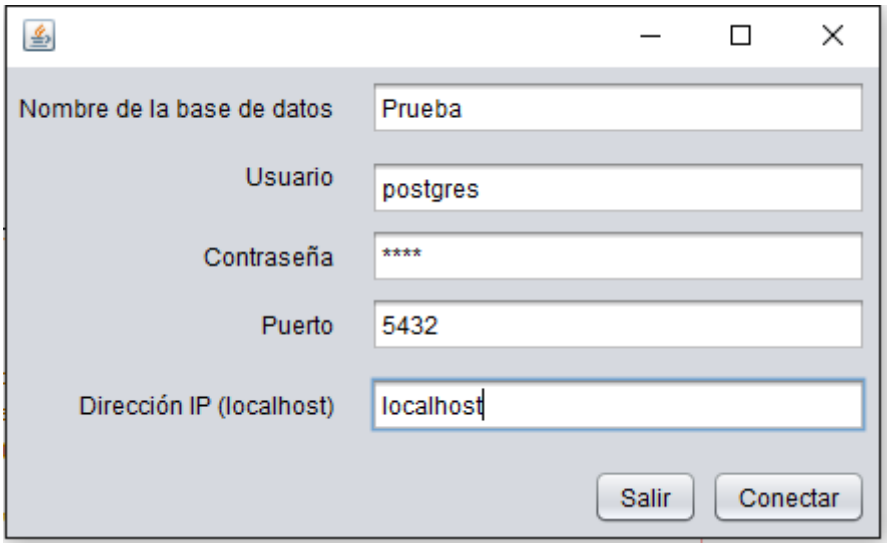


Figura 20. Muestra Conexión

Si se presiona el botón de Salir, la aplicación se cerrará por completo con todos sus procesos. Cuando se presione el botón de *Conectar,* se realizará la conexión y se abrirá una nueva ventana, en la cual se introducirá la consulta en el lenguaje de álgebra relacional. Para ello se ha creado un sistema de botones como el que se ve a continuación y el cual se explicará. La ventana que se mostrará será igual a la siguiente:

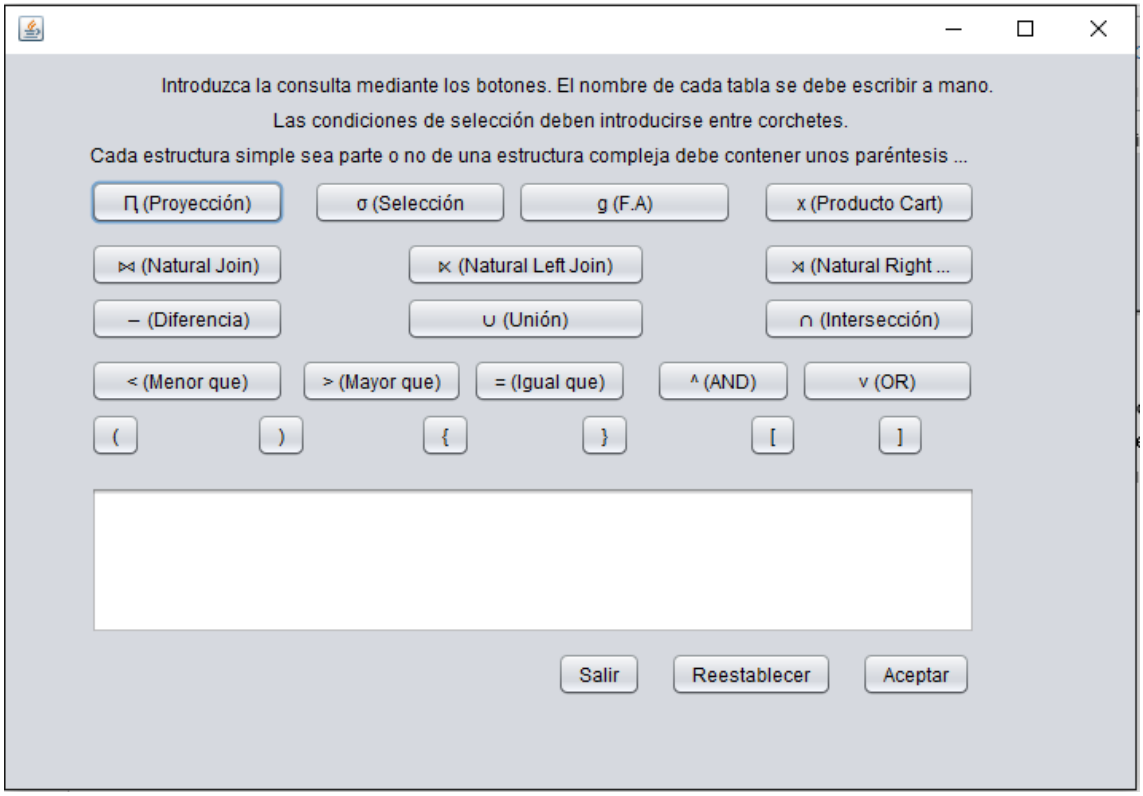


Figura 21. Pantalla Consulta

En esta ventana se pueden ver los operadores más importantes dentro del álgebra relacional, así como otros operadores matemáticos importantes para poder introducir cualquier consulta. La consulta irá apareciendo en el cuadro de texto a medida que se vaya introduciendo mediante teclado o botones. Si el usuario se ha equivocado y quiere borrar lo que lleva introducido hasta el momento solo tiene que pulsar el botón de *Reestablecer* y el cuadro de texto volverá a su estado original. Al igual que en el caso de la ventana anterior, si el usuario pulsa el botón de 'Salir' la aplicación se detendrá.

Como se ha comentado anteriormente, existen una serie de reglas que el usuario debe de tener en cuenta a la hora de introducir consultas. Estas están relacionadas con el uso de corchetes y paréntesis para poder permitir el funcionamiento de la aplicación. Esto es debido a que la aplicación trabaja con la consulta a partir de un sistema de corchetes y paréntesis generado por la misma consulta. Por lo tanto el usuario debe introducir todos los corchetes y paréntesis que sean necesarios para permitir su funcionamiento.

A continuación daré una serie de reglas generales que deben seguirse a la hora de crear una consulta con ejemplos representativos.

- Las condiciones de selección y proyección deben introducirse entre llaves: Cuando se quiera añadir una proyección o una selección en nuestra consulta las condiciones, deben de ir englobadas entre llaves "{}". Es una condición simple pero necesaria para que la aplicación pueda comprender dichos operadores.
  Algún ejemplo del uso de estas llaves se puede observar en las siguientes consultas:

  - (Π{a}(σ{t=1}(r)))
  - (Π{c}(σ{d=2}(s)))

  Como se puede ver en el primer ejemplo, la condición de selección "t=1" y la condición de proyección "a" deben de ir englobadas en llaves "{}" para un correcto funcionamiento de la aplicación.

- Cada estructura simple sea parte o no de una estructura compleja debe contener unos paréntesis que la engloben: Esto significa que cada consulta simple debe de ir entre paréntesis, ya sea una consulta en sí o parte de una consulta compleja. En el caso de consultas complejas del tipo "X union Y except Z" cada una de las consultas simples (X, Y, Z) deben de ir englobadas entre paréntesis pero no es necesario que la consulta global vaya englobado por los mismos.
  Algún ejemplo del uso de estos paréntesis se puede observar en las siguientes consultas:

  -(Π{t}((a)⋈(b)))
  -(Π{r,z}(σ{t=1}((a)∪(b))))
  -(Π{r}(σ{t=1}(a)))∪(Π{s}(σ{x=2}(b)))∪(Π{v}(σ{f=3}(c)))

En estas consultas se puede ver lo anteriormente comentado. Las dos primeras llevan cada una sus paréntesis interiores necesarios y aparte llevan unos paréntesis que las engloban. En el tercer caso se puede observar como cada consulta simple ("(Π{r}(σ{t=1}(a)))") lleva sus correspondientes paréntesis mientras que la consulta global no va englobada entre paréntesis.

- La condición de agregación (g) tiene sus propias reglas: Las condiciones para la función de agregación deben de ir englobadas entre llaves "{}" al igual que al tratar con proyecciones y selecciones. La diferencia llega a la hora de indicar la columna sobre la que queremos aplicar la función de agregado. Esta columna debe aparecer entre corchetes "[]" para un correcto funcionamiento de la aplicación.
  A continuación se muestra un ejemplo del uso de la función de agregación:

  -({t}g{sum[t]}(a))

  - (Π{t}(σ{t=1}({t}g{count[s]}(a))))

  Como se puede ver en el ejemplo las condiciones de agregación están contenidas entre llaves ({t}, {sum[t]), mientras que la columna sobre la que queremos aplicar la función de agregado aparece englobada entre corchetes ([t]).

- La consulta no debe terminar con un punto y coma: Pese a que en el álgebra relacional no es necesario terminar la consulta con un punto y coma, en el lenguaje SQL sí que es necesario. En este caso es la propia aplicación la que se ocupa de adjuntar un punto y coma al final de la expresión.

Una vez definidas las reglas para introducir una consulta correctamente el siguiente paso es introducir una consulta cualquiera y ejecutarla sobre nuestra base de datos.

En la siguiente imagen se puede ver una consulta, de las que antes hemos visto en las reglas, sobre la ventana de la consulta para después ejecutarla:

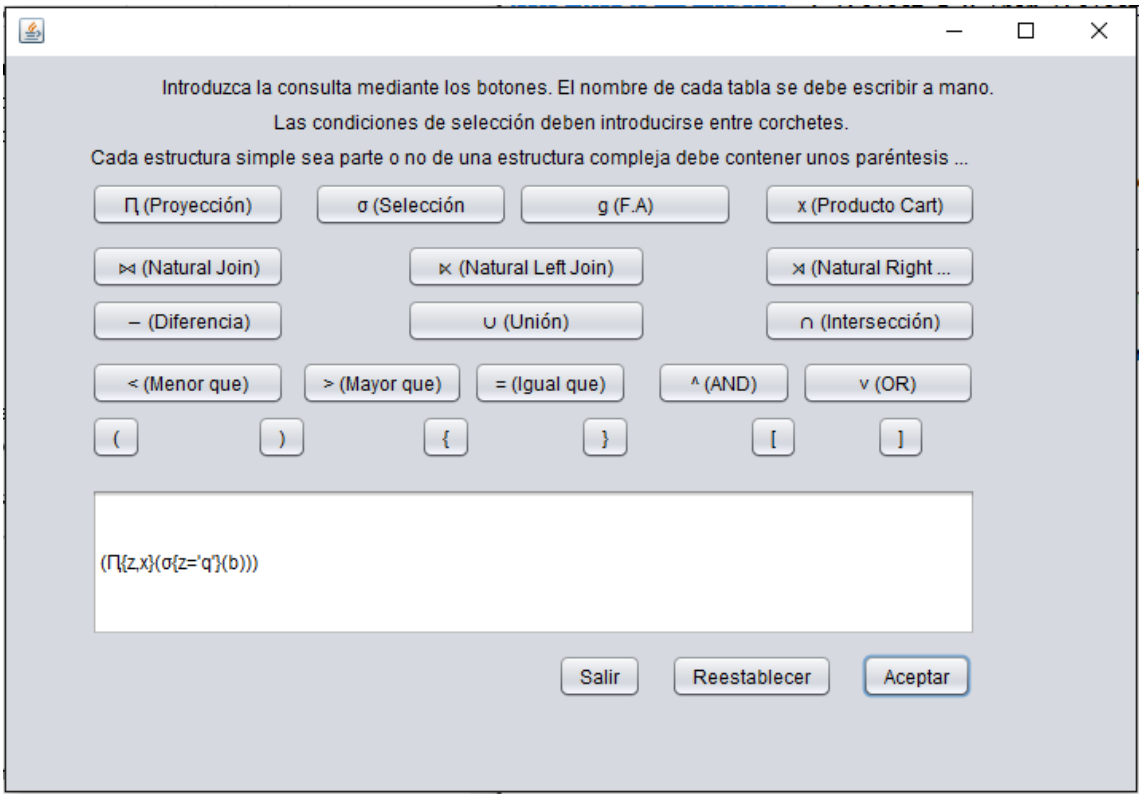


Figura 22. Muestra consulta

Pulsar el botón de 'Aceptar' para ejecutar la consulta y aparecerá la siguiente ventana de resultados, en la cual aparecerán los resultados que el sistema nos devuelve a partir de la consulta introducida traducida:

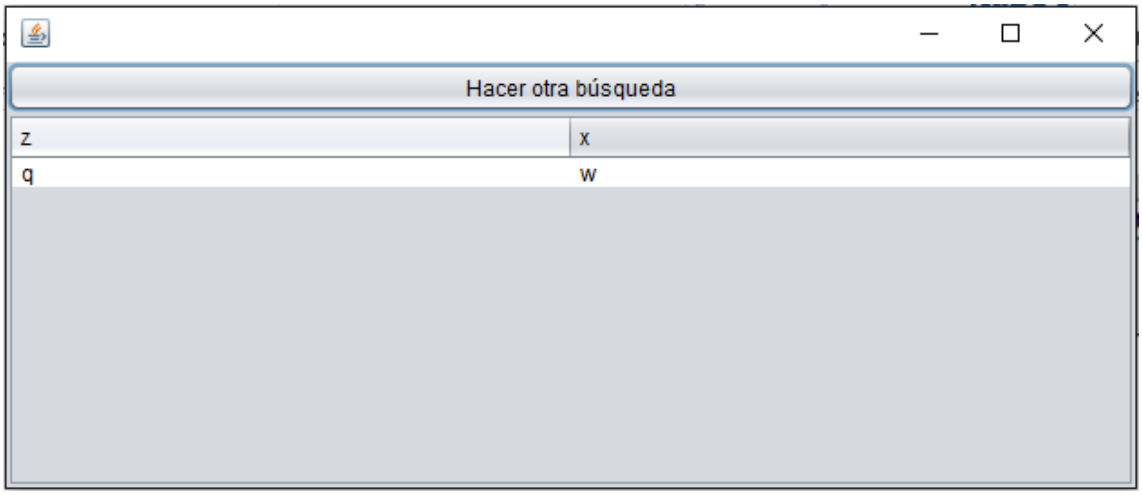


Figura 23. Muestra Resultados

Se puede ver en la ventana como el nombre de las columnas aparecen sobre un recuadro sombreado mientras que los valores de dichas columnas aparecen sobre fondo blanco. Esta ventana cuenta con la función de 'Hacer otra búsqueda'. Si se pincha sobre dicho botón se volverá a la ventana anterior para poder realizar otra consulta sobre la misma base de datos. La conexión con la base de datos se queda almacenada por lo que no hará falta introducir de nuevo los credenciales de autentificación.

Por lo tanto se realizarán dos ejemplos más de búsquedas para mostrar cómo funciona la aplicación frente a distintas consultas. La primera será la siguiente consulta ((Π{r}(a))⋈(Π{z}(b))). Lo primero es introducirla mediante los botones disponibles en la pantalla que aparece una vez que hayamos pulsado en el botón 'Hacer otra búsqueda'.

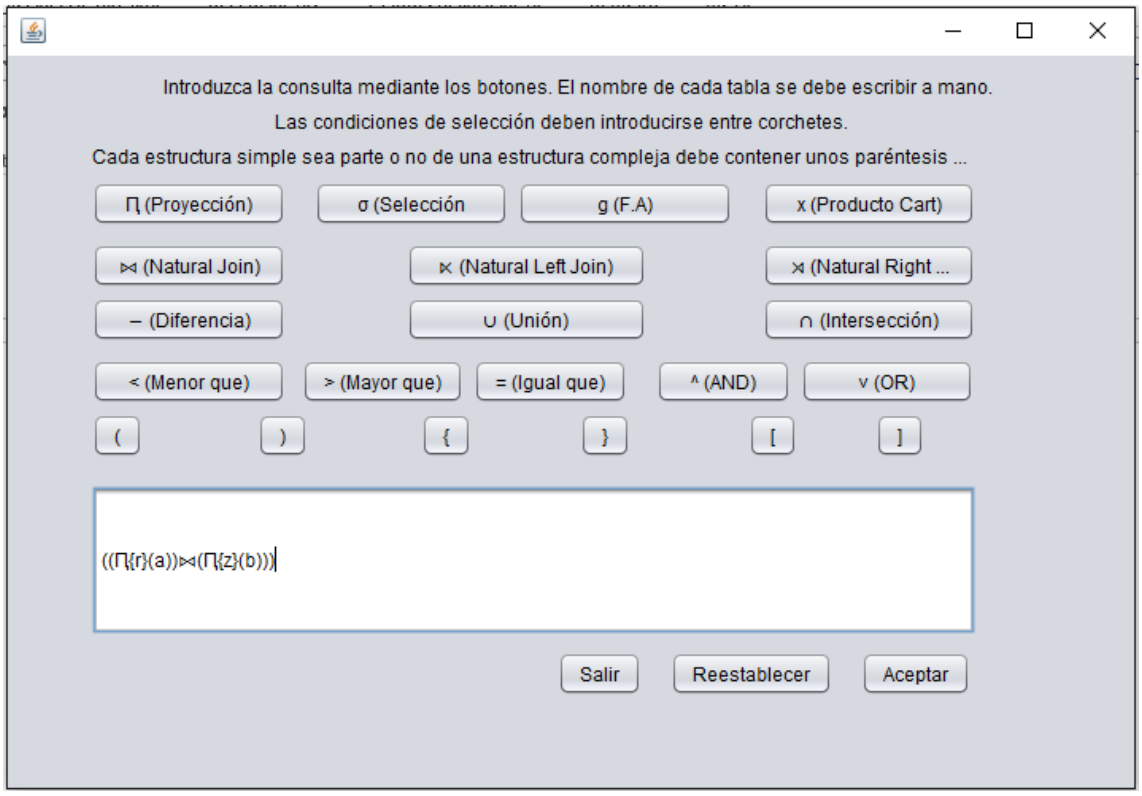


Figura 24. Muestra consulta 2

En la siguiente imagen podemos ver mejor el uso de paréntesis que se ha comentado antes. Cada tabla y cada subconsulta están englobados en su par de paréntesis correspondiente, así como la consulta total, la cual va también englobada con su par de paréntesis necesario. El resultado de la búsqueda una vez pulsamos el botón de 'Aceptar' sería el siguiente.

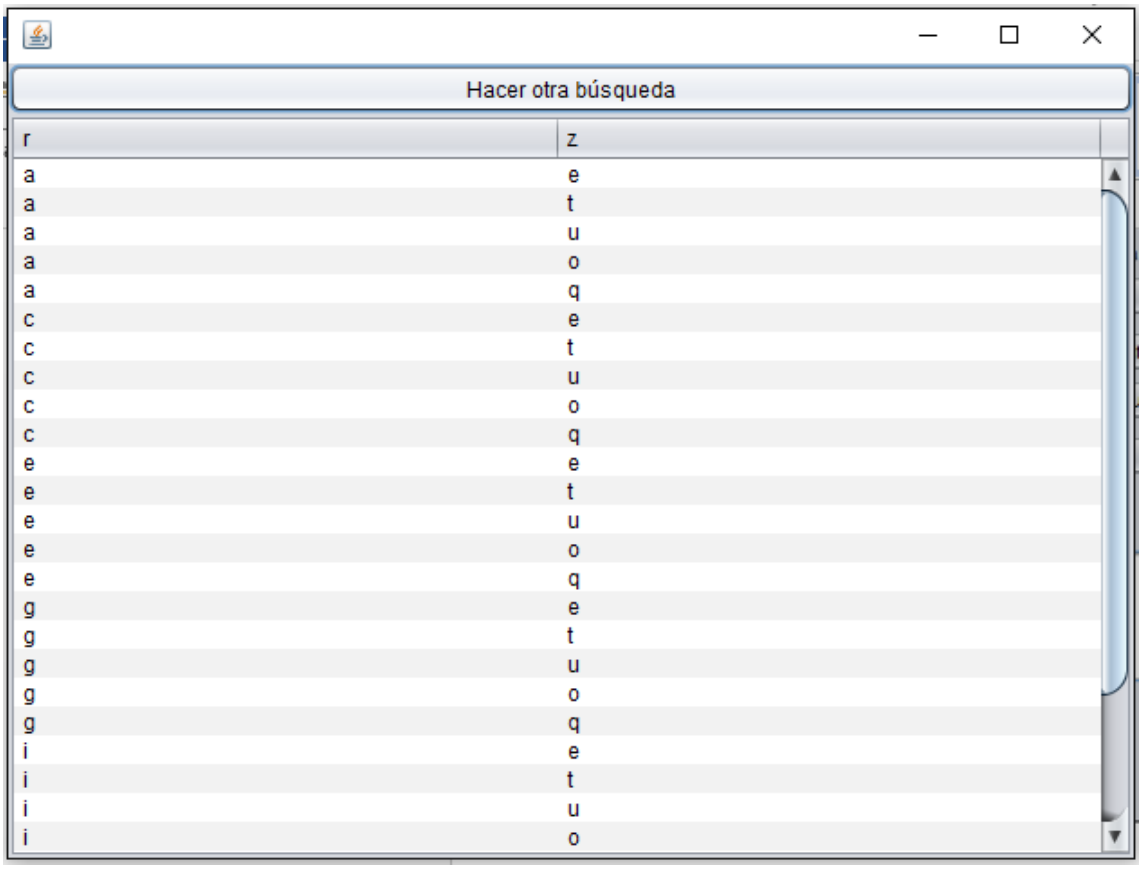


Figura 25. Muestra Resultados 2

Como se ha realizado un natural join sin criterios de selección se puede ver la reunión natural entre las tablas mencionadas, mostrando las columnas que han sido definidas.

La última consulta la haré sobre la base de datos creada en el apartado 'TESTING', siendo esta una base de datos más intuitiva. Al ser una base de datos distinta a la que ya se está conectado deberemos salir de la aplicación y volver a ejecutarla.

No se mostrará otra vez la pantalla 'Inicio' en la que se deben introducir los datos de la nueva conexión. Una vez hecho esto se inserta la nueva consulta sobre la pantalla dedicada a ello. Quedaría de la siguiente forma:

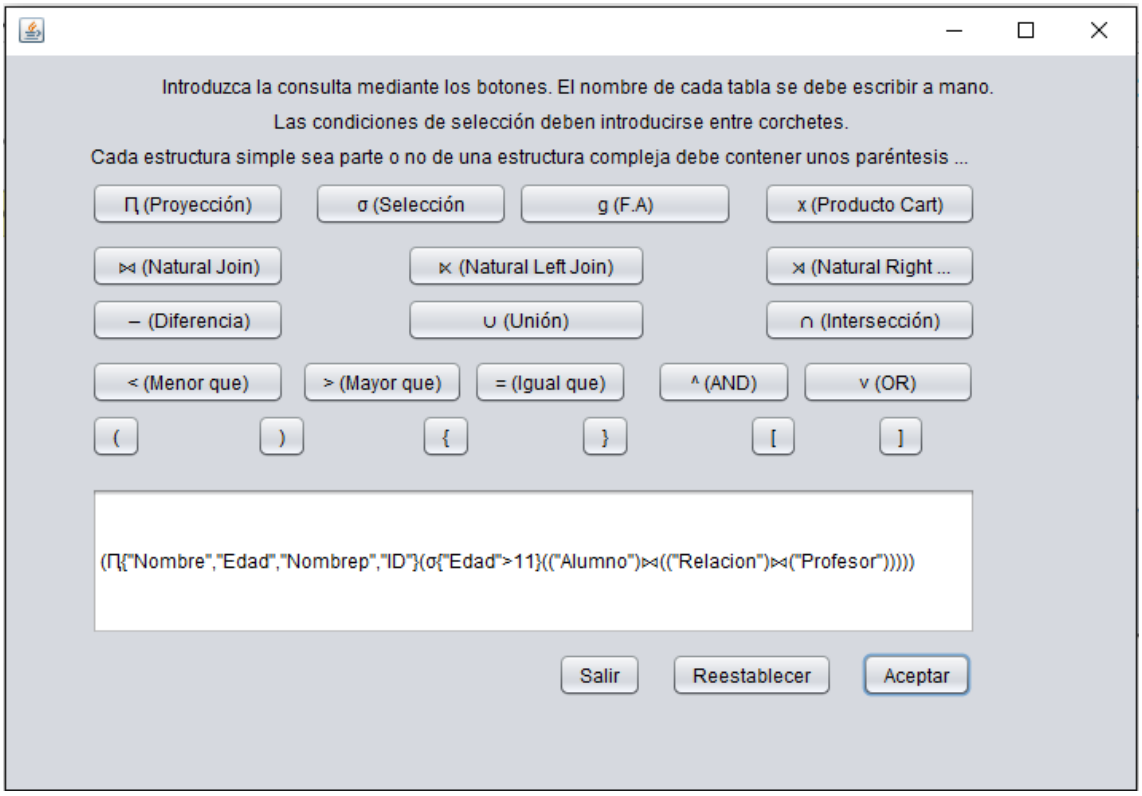


Figura 26. Muestra consulta 3

Se pulsa en el botón 'Aceptar' y se pueden ver los resultados que aparecen.

Nombre	Edad	Nombrep	ID
Ivan	12	Juan	id1
Natalia	13	Daniel	id3
Natalia	13	Jaime	id2
Diego	12	Ana	id1
Diego	12	Carlos	id1
Julia	13	Daniel	id3
Julia	13	Juan	id1
Ana	12	Daniel	id3

Figura 27. Muestra Resultados 3

Con esto termino el Manual de Usuario, en el cual he podido mostrar cómo utilizar la aplicación, ejemplos de uso y las directrices que hay que seguir para introducir una consulta correctamente.

Presupuesto

A continuación voy a detallar los costes de la realización del proyecto. Para ello separaré los costes de hardware, software y humanos.

Costes hardware

Para este apartado se incluirán todos los elementos hardware que hayan sido necesarios adquirir para poder realizar el proyecto. En este caso se adquirió un portátil, ya que el anterior que tenía era bastante viejo, y un ratón inalámbrico para facilitar la tarea. El portátil en cuestión es un Lenovo Ideapad 100-15IBD y el ratón es un Logitech M185. Con estos datos podemos hacer una tabla mostrando los costes.

CONCEPTO	PRECIO
Lenovo Ideapad 100-15IBD	459€
Logitech M185	25€
COSTE TOTAL	484€

Tabla 1. Costes Hardware

Como se puede ver en la tabla los costes hardware ascienden a 484€.

Costes software

En este apartado se incluyen todos los costes ocasionados de aplicaciones software adquiridas a lo largo del proceso. En este caso ninguna de las aplicaciones ha tenido coste alguno ya que se ha hecho uso de aplicaciones open source o gratuitas, como es el caso de Netbeans o de PostgreSQL. Por lo tanto los costes software de la aplicación son de 0€.

Costes humanos

En este apartado contaré los costes humanos que ha tenido el proyecto, que en este caso sería mi salario si se hubiese tratado de un trabajo remunerado. Para ello, y al no tener experiencia, me asigno una remuneración de 5,5 €/hora, que es aproximadamente lo que cobra un becario al principio de entrar a una empresa. A partir de ahí, y según un cálculo aproximado de las horas que he dedicado, obtendré el coste humano que esta aplicación ha ocasionado.

HORAS	COSTE POR HORA	COSTE TOTAL
150 horas	5,5 €/hora	825€

Tabla 2. Costes Humanos

Como se puede ver en la tabla los costes humanos ascienden a 825€.

Coste total

En este apartado recopilaremos los distintos conceptos para lograr el coste total del proyecto. Para ello expondré una tabla en la que muestro los distintos costes asociados.

Concepto		Coste
Costes Hardware		
	Lenovo Ideapad 100-15IBD	459€
	Logitech M185	25€
Costes software		
	Ninguno	0€
Costes humanos		
	Remuneración	825€
COSTE TOTAL		1308€

Tabla 3. Costes totales

El coste total del proyecto asciende a 1308€.

Mejoras posibles

Una vez desarrollada la aplicación surgen una serie de mejoras que se podrían realizar para lograr un programa más completo y con mayores posibilidades.

- Conexión con otros SGBD: Actualmente la aplicación solo puede conectarse con cualquier base de datos tratada mediante PostgreSQL. Una posible mejora sería modificarla para que pueda conectarse con otros sistemas gestores como pueden ser MySQL, Oracle, Microsoft SQL Server o Microsoft Access. Sería una mejora interesante y que no ocuparía una gran cantidad de esfuerzo para desarrollarla.

- Ampliación de operadores: La aplicación cuenta con todos los operadores más importantes del álgebra relacional, pero no con todos los que existen. Después de comentarlo con el tutor, concluimos cuales serían todos los operadores que iban a aparecer, pero no son todos ya que hay una gran cantidad de operadores. Aparte de esto, hay muchos de ellos que no se tratan a lo largo del curso académico, por lo que era innecesario tratarlos. Sería una mejora interesante si se piensa utilizar esta aplicación para funcionalidades distintas a las previstas. En este caso sería una mejora que sí ocuparía una cantidad importante de esfuerzo ya que habría que configurar el comportamiento de cada operando lo que no es una tarea fácil ni rápida.

- Optimización de código: Como casi cualquier código complejo existente, siempre se puede optimizar para que necesite de menos recursos de la máquina. A lo largo del desarrollo del programa he tenido en cuenta este aspecto evitando bucles, llamadas y sentencias innecesarias. Pero aun así el código puede optimizarse un poco más con el uso de técnicas y patrones especializados. Pese a ello, debido a que la carga que la aplicación supone al ordenador no es mucha, no creo que fuese una mejora muy interesante comparando el esfuerzo que supone y el impacto que ocasiona.

Conclusiones

Por último expondré las conclusiones que me han generado al desarrollar este Proyecto de Fin de Grado. Desde el principio y hasta el final me ha parecido un Proyecto muy interesante, ya que al tener un uso posterior por nuevos alumnos siempre es una motivación extra. Saber que tu trabajo va a conseguir facilitar el aprendizaje de otras personas es una fuente de motivación importante.

Otro de los aspectos que me gustaría resaltar es que a lo largo de su realización he utilizado dos de las partes más importantes del Grado, como es la Programación y las Bases de Datos. Para su desarrollo he necesitado utilizar mucho del material recogido a lo largo de estos años, por lo que es una satisfacción poder ver como lo aprendido a lo largo del camino tiene una finalidad práctica. Para su realización he necesitado hacer uso de apuntes, ejercicios y prácticas realizadas en distintas asignaturas. Esto me ha hecho además poder recordar aspectos que tenía más olvidados y reforzar partes que no se habían tratado con tanta importancia.

Otra cosa que quiero añadir ha sido la dificultad que me ha ocasionado su realización. Cuando elegí desarrollar este proyecto sabía que no iba a ser algo sencillo, pero a lo largo del trayecto me han surgido muchos más problemas e inconvenientes de los que inicialmente supuse. Pero esto no lo veo como algo negativo, si no que me ha ayudado a esforzarme más y a saber lidiar con estos problemas y saber que no debo rendirme ante escenarios como este.

Por otro lado nunca había desarrollado una aplicación de estas magnitudes. En el ámbito de la Programación lo máximo que había desarrollado eran las aplicaciones que solicitaban las asignaturas, por lo que nunca eran de un tamaño como la que acabo de realizar. Eso me ha hecho hacerme una idea de lo que puede ser desarrollar una aplicación importante y de lo que me puede deparar una vez entre en el mundo laboral.

Por último creo que se han conseguido todos los objetivos que se planeaban inicialmente por lo que pienso que la realización de este Proyecto ha sido un éxito y espero que sea de ayuda para las nuevas generaciones de Informática de la UAH.

Anexos

Bibliografía

Abraham Silberschatz, Henry F.Korth y S. Sudarshan (2002). *Fundamentos de bases de datos* (4º edición) McGraw-Hill/InterAmericana de España, S.A.U.

Apuntes teoría de las asignaturas *Bases de Datos* y *Bases de Datos Avanzadas.*

https://jdbc.postgresql.org/documentation/head/connect.html

http://stackoverflow.com/

https://docs.oracle.com/javase/7/docs/api/